



Programmieren mit Pfiff

Alle wichtigen Speicherstellen

- ★ Kompletter Kurs »Memory-Map«
- ★ Die interessantesten ROM-Routinen für Basic-Programmierer
- ★ Die wichtigsten PEEKs & POKEs für Grafik, Musik und Schnittstellen

Neue heiße Tips & Tricks für C64 und C128

- ★ Schnelle Rechenroutinen
- ★ Multitasking für Basic-Programme

Viele nützliche Listings zum Abtippen

Alle Programme auf Diskette erhältlich



Der Hunger nach Wissen



Es ist schon faszinierend, was man dem C 64 alles entlocken kann. Ich kann mich noch an die Zeit erinnern, als der C 64 auf den Markt kam. Der Eindruck, den die ersten Anzeigen und Testberichte hinterließen, war überwältigend. Eine völlig neue Dimension tat sich auf: Grafik mit 320 mal 200 Punkten, sagenhafte 64 KByte Speicher, Sprites und umfangreiche Soundmöglichkeiten ließen einem das Wasser im Mund zusammenlaufen. Aber erst der bis dahin unerreicht niedrige Preis von knappen 1500 Mark (der schnell auf 1300 Mark abrutschte) ließ die innere Kaufhürde schmelzen. Als der C 64 dann auf dem Tisch stand, begann jedoch schon das Dilemma. Das Basic V2.0 war erschreckend spartanisch. Keine Befehle für Grafik, keine für Musik. Das Handbuch half zwar über die ersten Hürden, warf dann jedoch mehr Fragen auf als Antworten zu geben. Software gab es kaum, und die angebotenen Programme waren zu teuer.

Das war die Zeit der vielen Fragen und wenigen Antworten. Die ersten großen Entdeckungsreisen durch den C 64 begannen. Mit einer vorher kaum vorstellbaren Energie und Ausdauer wurde probiert, getüftelt und experimentiert. Fiel während eines Bummels durch ein Kaufhaus das Wort »C 64« oder »Commodore«, wurden die Ohren zu hochsensiblen Empfangsantennen mit kolossaler Richtwirkung. Jede noch so geringe Information wurde aufgesogen, wie wenn ein Ertrinkender nach Luft schnappt. Die Hoffnung auf eine neue POKE- oder PEEK-Adresse ersetzte glatt ein Mittagessen und machte die Nacht zum Tage.

Das war die Zeit des Fiebers nach Informationen. Es gab am Anfang ja noch nichts. Kein 64'er-Magazin, keine Literatur. So war es überhaupt kein Wunder, daß das erste Buch zum C 64 ein Renner wurde. »C 64 - Intern« war wie eine Offenbarung. Zum erstenmal konnte der große Hunger gestillt werden.

Das war die Zeit, in der man sich zurückzog, las, bis einem die Augen tränten.

Der C 64 wurde innerhalb kürzester Zeit zum Verkaufsrenner. Nach und nach gab es auch mehr Literatur. Die Computer-Zeitschriften reservierten immer mehr Platz für den C 64. Kurz nachdem er zum meistverkauften Heimcomputer Deutschlands wurde, kam das 64'er-Magazin. Leser die den C 64 schon länger kannten, gaben alles was sie wußten. Sie schrieben Grundlagenartikel, Testberichte, Kurse. Endlich bekam man jeden Monat knallharte Informationen. Von Profis für Anfänger, Fortgeschrittene und Kenner. Jetzt begann die Zeit, in der man aus dem Vollen schöpfen konnte.

Vor allem die Kurse waren und sind so beliebt, daß für viele Leser lange Zeit kein Ende in Sicht war, zum Beispiel der Assembler-Kurs oder die Memory-Map. Einen Nachteil hat das Ganze aber. Sucht man eine bestimmte Information, müssen, wie zum Beispiel bei der Memory-Map, bis zu 18 64'er-Ausgaben durchsucht werden. Hier erfüllen die 64'er-Sonderhefte eine wichtige Funktion. Stoff, der sich über lange Zeit angesammelt hat, kann in einem Sonderheft zusammenhängend veröffentlicht werden. Zusammen mit neuen Listings, hervorragenden Tips & Tricks und vielen Grundlageninformationen wird in jedem Sonderheft ein Schwerpunktthema behandelt.

In diesem Sonderheft finden Sie den kompletten Kurs »Memory-Map mit Wandervorschlägen«, der im 64'er, Ausgabe 11/84 begann und mit Ausgabe 6/86 abgeschlossen wurde. Er behandelt die wichtigsten Speicherzellen (Adressen) des C 64 und VC 20, erklärt deren Funktion und fordert anhand vieler Beispiele zum Mitmachen auf. Die ersten tausend Byte des C 64 kann man getrost als sein Kurzzeitgedächtnis bezeichnen. Mit dem Wissen über die Wirkung dieser Zellen lernen Sie nicht nur den Computer besser verstehen, sondern sind auch in der Lage, ihn zu beherrschen. Die »Memory-Map mit Wandervorschlägen« ist ein umfangreiches Nachschlagewerk, geschrieben für Einsteiger, Fortgeschrittene und Wißbegierige.

C 128-Besitzer bekommen vielleicht einige Probleme, weil nicht alle in der Memory-Map beschriebenen Adressen für den C 128 gelten. Deshalb gibt es einen eigenen Artikel, in dem die Unterschiede zum C 64 erklärt und beschrieben werden. Wer sich mit Maschinensprache beschäftigt, ist noch mehr auf die Erklärung der verschiedenen Speicherstellen angewiesen als der Basic-Programmierer. Als Assemblerprogrammierer arbeitet man sozusagen nur mit »PEEKs & POKEs«. Aus diesem Grund haben wir einen Leckerbissen für ihn parat: Ausführlich und leicht verständlich wird erklärt, wie Zahlen und Zeichen ein- und ausgegeben werden, wie man mit ihnen rechnet, in andere Formate umwandelt, etc. Mathematik in Assembler ist kein leichtes Thema. Aber mit den Betriebssystemroutinen des C 64 und einer guten Übersicht wird sie zum Kinderspiel. Doch auch überzeugte Basic-Programmierer können Systemroutinen benutzen und vorteilhaft anwenden. Was Sie beachten müssen, welche Tricks Sie kennen sollten, verrät Ihnen ein eigener Artikel. Natürlich finden Sie zusätzlich wieder viele Listings zum Abtippen und vor allem jede Menge PEEKs & POKEs und kurze und interessante Tips & Tricks.

Betrachtet man die heutige Situation und vergleicht sie mit der damaligen, oben geschilderten, so hat sich nur eines geändert: die verfügbare Information. Geblieben ist der Hunger nach Wissen, Grundlagen und Programmen. Und von daher kann ich jeden neuen C 64-Besitzer nur beneiden.

(Georg Klinge)

PROGRAMM-SERVICE



Bestellungen in der Schweiz: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Tel. 042/41 56 56
Bestellungen in Österreich: Bücherzentrum Meidling, Schönbrunner Straße 261, A-1120 Wien, Tel. 0222/83 31 96,
Microcomput-ique E. Schiller, Fasangasse 21, A-1030 Wien, Tel. 0222/78 56 61,
Ueberreuter Media Handels- und Verlagsgesellschaft mbH, Alser Straße 24, A-1091 Wien, Tel. 0222/48 15 38-0
Bestellungen aus anderen Ländern bitte per Auslandspostanweisung!

Das Angebot dieser Ausgabe:

Wer keine Zeit oder Lust hat, alle Programme selbst in mühevoller Kleinarbeit abzuschreiben, kann wieder auf den bewährten Programm-Service zurückgreifen. Alle Programme, die mit dem Diskettensymbol ▪ im Inhaltsverzeichnis gekennzeichnet sind, gibt's auf Diskette.

1 Diskette
Bestell-Nr. L6 86 S7D

DM 29,90*
(sFr. 24,90/öS 299,-*)

* inkl. MwSt. Unverbindliche Preisempfehlung

Programme aus früheren Ausgaben:

64'er-Ausgabe 6/86
Bestell-Nr. L6 86 06D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

- Prodisk (AdM) – Eine professionelle Diskettenverwaltung — S. 50
- Master-Text (LdM) – Die beste Textverarbeitung zum Abtippen — S. 55
- Etiketten (Basic und compilierte Version) – Professionelle Etiketten für Epson-Drucker und Kompatible — S. 69
- Erweiterung zu Pseudo-Scroll (3/86) — S. 77
- Zahlen eingeben mit dem Joystick — S. 77
- Grafik-Erweiterung für Lores-Bildschirm — S. 79
- Garbage-Collection-Anzeige (mit Beispiel) — S. 79
- 43007 statt 38911 Basic-Bytes für C64 durch genialen Trick — S. 80
- Eine sinnvolle Anwendung der FN-Anweisung — S. 82
- Super-Autostart — S. 82
- Undim. Var. Dump (Ausgabe der nicht-DIMensionierten, nur für C128, Variablen) — S. 83
- F. Key-Display (vier zusätzliche Bildschirmzeilen, nur für C128, zeigen die Funktionstastenbelegung) — S. 83
- Find (Basic-Erweiterung für das Basic 7.0 des C128) — S. 84
- Flashmove (C64-Programm schneller laden) C128 mit Floppy 1571 — S. 85
- Sprites invertieren (C128) — S. 85
- Basic-Tool (vier zusätzliche Basic-Befehle für C16) — S. 86
- Wahl-Cursor — S. 90
- Hypra-Ass mit Datasette (Erweiterung) — S. 95
- Von Basic zu Assembler (11 Listings) — S. 134
- Shopmaster (konvertiert Printshop-Grafik zu Printmaster-Grafik) — S. 101
- Read Vizawrite und Vi-Co-CC — S. 163
- Shades und Synth Dive (zwei Super-Musikstücke) — S. 173

64'er-Ausgabe 5/86
Bestell-Nr. L6 86 05D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

- 64er DOS V3 — S. 10
- Grafik und Computeranimation — S. 18
- Fantastische Grafik — S. 29
- Disk-Wizard (LdM) — S. 54
- Super Hardcopies für Epson-Drucker und Kompatible — S. 63
- Greatprint – Große Zeichen auf dem Bildschirm (mit Demo) — S. 69
- Super Hardcopy (Epson, 1520, CP 80X) — S. 70
- Der »Epson-Plotter« — S. 79
- Charakter-Editor — S. 81
- Steel-Slab (Spielelisting) — S. 86
- Tips&Tricks zum C128
- Merge — S. 95
- Spriteslow — S. 97
- Old — S. 98
- Eingabe — S. 98
- Tips&Tricks für Profis
- Alle Pokes — S. 99
- Outadr — S. 100
- Array-Sort — S. 100
- Basic-Programme im Interrupt — S. 103
- Neue Module für Hypra-Basic — S. 103
- Pascal-Kurs Zeichen — S. 142
- Joseph — S. 142
- Matrimult – ein Programm zur Multiplikation beliebiger Matrizen — S. 145
- Adreßprogramm mit Superbase 64 — S. 168
- Zviza – Zeichensatz für Vizawrite — S. 171

64'er-Ausgabe 4/86
Bestell-Nr. L6 86 04D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

- Quizmaster — S. 53
- Hypra Basic — S. 58
- Druckroutine zu DATABASE (DB II) — S. 63
- Hardmaker — S. 67
- Synchro Justage — S. 77
- Micro-Tagebuch — S. 77
- Ex-Line — S. 78
- Soft-Flash – Trick an der Floppy — S. 79
- Strich-Cursor — S. 79
- Upside Down – Dreht den Bildschirm um 180 Grad — S. 79
- Disk-Optimizer – Basic und Compilerversion — S. 80
- Apfelmännchen — S. 84
- Autochange – Ihr C128 springt automatisch in den richtigen Modus — S. 85
- Taktzyklen – Für Basic und Maschinenprogramme — S. 86

64'er-Ausgabe 3/86
Bestell-Nr. L6 86 03D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 2/86
Bestell-Nr. L6 86 02D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 1/86
Bestell-Nr. L6 86 01D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 12/85
Bestell-Nr. L6 85 12D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)
Bestell-Nr. L6 85 12K Kassette
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 11/85
Bestell-Nr. L6 85 11A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 10/85
Bestell-Nr. L6 85 10A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 9/85
Bestell-Nr. L6 85 09A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 8/85
Bestell-Nr. L6 85 08A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 7/85
Bestell-Nr. L6 85 07A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 6/85
Bestell-Nr. L6 85 06A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 5/85
Bestell-Nr. L6 85 05A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 4/85
Bestell-Nr. L6 85 04A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 3/85
Bestell-Nr. L6 85 03A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 2/85
Bestell-Nr. L6 85 02A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Ausgabe 1/85
Bestell-Nr. L6 85 01A
DM 29,90* (sFr. 24,90/öS 299,-*)

64'er-Sonderhefte

Sonderheft 6/86 – Grafik
2 Disketten mit allen Programmen
Bestell-Nr. L6 86 S6D1
DM 34,90* (sFr. 29,50/öS 349,-*)
1 Diskette mit Giga-CAD-Demos
Bestell-Nr. L6 86 S6D2
DM 19,90* (sFr. 17,-/öS 199,-*)
3 Disketten mit allen Programmen und Demos
Bestell-Nr. L6 86 S6D3
DM 49,80* (sFr. 43,50/öS 498,-*)

Sonderheft 5/86 – Grundwissen
Bestell-Nr. L6 86 S5D 1 Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

Sonderheft 4/86 – Abenteuer
Bestell-Nr. L6 86 S4D 2 Disketten
DM 34,90* (sFr. 29,50/öS 349,-*)

Sonderheft 3/86 – C16, C116, VC20, Plus 4
1 Diskette für VC20 und C16/116:
Bestell-Nr. L6 86 S3 CD
DM 29,90* (sFr. 24,90/öS 299,-*)
1 Kassette für VC20:
Bestell-Nr. L6 86 S3 KV
DM 19,90* (sFr. 17,-/öS 199,-*)
1 Kassette für C16:
Bestell-Nr. L6 86 S3 KC
DM 19,90* (sFr. 17,-/öS 199,-*)

Sonderheft 2/86 – Tips&Tricks
Bestell-Nr. L6 86 S2D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

Sonderheft 1/86 – C128er
Bestell-Nr. L6 86 S1D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)

Sonderheft 8/85 – Assembler
Bestell-Nr. L6 85 S8D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)
Bestell-Nr. L6 85 S8K Kassette
DM 19,90* (sFr. 17,-/öS 199,-*)

Sonderheft 7/85 – Professionelle Anwendungen
Bestell-Nr. L6 85 S7D 2 Disketten
DM 34,90* (sFr. 29,50/öS 349,-*)
Bestell-Nr. L6 85 S7K 4 Kassetten
DM 34,90* (sFr. 29,50/öS 349,-*)

Sonderheft 6/85 – Top-Themen
Bestell-Nr. L6 85 S6 2 Disketten
DM 34,90* (sFr. 29,50/öS 349,-*)

Sonderheft 5/85 – Floppy, Datasette
Bestell-Nr. L6 85 S5D Diskette
DM 29,90* (sFr. 24,90/öS 299,-*)
Bestell-Nr. L6 85 S5K Kassette
DM 19,90* (sFr. 17,-/öS 199,-*)

Sonderheft 4/85 – Grafik
Bestell-Nr. L6 85 S4A
DM 29,90* (sFr. 24,90/öS 299,-*)

Sonderheft 3/85 – Spiele
Bestell-Nr. L6 85 S3 A 2 Disketten
DM 34,90* (sFr. 29,50/öS 349,-*)

Sonderheft 2/85 – Abenteuerspiele
Bestell-Nr. L6 85 S2
DM 34,90* (sFr. 29,50/öS 349,-*)

Sonderheft 1/85 – Tips&Tricks (2. überarb. Auflage)
Bestell-Nr. CB 023 Floppy-Utilities
DM 29,90* (sFr. 24,90/öS 299,-*)
Bestell-Nr. CB 024 Hilfsprogramme
DM 29,90* (sFr. 24,90/öS 299,-*)

* inkl. MwSt. Unverbindliche Preisempfehlung.

Bitte verwenden Sie für Ihre Bestellung und Überweisung die eingeheftete Postgiro-Zahlkarte, oder senden Sie uns einen Verrechnungs-Scheck mit Ihrer Bestellung.
Sie erleichtern uns die Auftragsabwicklung, und dafür berechnen wir Ihnen keine Versandkosten.

MEMORY MAP mit Wandervorschlägen

Es steckt sehr viel im C64. Wir werden Ihnen im Rahmen dieses Kurses die Bedeutung und Anwendung der Speicher und Register von Betriebssystem und Interpreter näherbringen.

Dieser Kurs ist eine zusammenhängende Wiedergabe einer Serie von Aufsätzen, die im 64'er, Ausgabe 11, November 1984 angefangen hat und sich in 18 Folgen bis Ausgabe 6, Juni 1986, hinzog.

Die vorliegende Fassung hat profitiert von vielen Zuschriften, die ich im Laufe des Kurses erhalten habe, mit Kritik, Hinweisen auf Fehler, Fragen und Verbesserungsvorschlägen. Für diese Mitarbeit der Leser möchte ich mich an dieser Stelle recht herzlich bedanken.

Ein Inhaltsverzeichnis der Adresse ist wenig sinnvoll. Ich habe allerdings die Adressen in einer Liste zusammengefaßt, geordnet nach ihren Funktionen, nicht nach ihrer Reihenfolge.

Dadurch soll Ihnen eine weitere Möglichkeit zum Nachschlagen geboten werden.

Auflisten kann ich lediglich die Texteinschübe und die Tabellen, zur Übersicht für eilige Sucher.

Hinweise und Tips über nützliche PEEK- und POKE-Adressen gehören zum Standard-Repertoire einer Computer-Zeitschrift. Ebenso häufig werden Leserfragen zu diesem Thema gestellt, obwohl mehrere Handbücher für die beiden Heimcomputer von Commodore bereits Speicherlisten (auf englisch »Memory Map«) enthalten.

Warum ich mich jetzt auch noch mit diesem Thema befassen will, hat zwei Gründe. Zum einen stört mich, daß ein Hinweis wie:

»...mit POKE 19,1 läßt sich das Fragezeichen bei INPUT-Befehlen unterdrücken...«

zwar richtig und auch anwendbar ist, aber halt nicht erklärt, was da eigentlich passiert und welche Folgen das für ein Programm haben kann. Zum anderen vermisse ich speziell in den Speicherlisten nähere, auch für den Anfänger verständliche und irgendwann einmal verwertbare Angaben.

Ich habe mir deshalb vorgenommen, Ihnen die Bedeutung

Texteinschub Nr. 1
Der USR-Befehl

Hand aufs Herz: Haben Sie USR schon einmal benutzt? Ohne Zweifel gehört dieser Befehl zu den seltenen. Ich will ihn daher hier kurz erläutern. USR hat dieselbe Funktion wie SYS, nämlich aus einem Basic-Programm direkt in ein Maschinenprogramm zu springen und dort solange weiterzufahren, bis mit dem Befehl RTS (entspricht dem Basic-Befehl RETURN) in das Basic-Programm zurückgesprungen wird. Die Sprungadresse in das Maschinenprogramm steht bei SYS gleich hinter dem Befehl.

Bei USR muß die Adresse zuerst in die Speicherzellen 1 und 2 (aha!!) gePOKEt werden.

Beispiel – Sprung auf 56524 ($DCCC):

mit SYS: `SYS 56524`

mit USR: `POKE 1,204:POKE 2,220:X=USR(Y)`

Kein Wunder, daß USR selten benutzt wird. Aber erstens ist er durch das POKEn der Low-/High-Byte-Darstellung aufgebläht und zweitens hat er auch wesentlich mehr Fähigkeiten als SYS.

Sein Argument, im obigen Beispiel also das »Y«, wird nämlich zuerst in den »Fließkomma-Akkumulator« FAC 1 (Floating Point Accumulator Nr. 1) gebracht, der sich in den Speicherzellen 97 bis 102 ($61 bis $66) befindet. Da wir ihn auf unserer Reise durch den Speicher noch treffen werden, brauche ich jetzt nicht näher darauf einzugehen. Wichtig ist lediglich, daß der Wert von »Y« dann vom angesprungenen Maschinenprogramm verarbeitet werden kann. Das Resultat kommt dann wieder in diesen FAC 1 und steht als Wert von X (siehe Beispiel oben) dem Basic-Programm zur Verfügung.

Mit USR kann man also Variable ins Maschinenprogramm zur Bearbeitung und zurück transferieren – und das ist der Unterschied zum SYS-Befehl. Ich möchte das an einem kleinen Beispiel demonstrieren. Statt allerdings ein Maschinenprogramm selbst zu schreiben, verwende ich beziehungsweise springe ich auf eine Routine des Betriebssystems, welches Werte des FAC 1 für mathematische Operationen verwendet.

Als mathematische Operation wähle ich das eingebaute Programm für INT, welches im VC 20 ab Speicherzelle 56524 ($DCCC) steht, im C 64 steht es ab 48332 ($BCCC). Dieses wollen wir verwenden:

In Zeile 10 definieren wir einen Wert für die Variable X, der in das Maschinenprogramm gebracht werden soll. Mit Zeile 20 bringen wir die Startadresse des Maschinenprogramms in die Speicherzellen 1 und 2.

Laut Kochrezept teilen wir die Adresse 56524 auf in ein Low-Byte = 204 und ein High-Byte = 220.

Der Befehl in Zeile 30 löst den ganzen USR-Vorgang aus, Zeile 40 gibt uns das Resultat.

```
10 Y=14.35
20 POKE 1,204:POKE 2,220
30 X=USR(Y)
40 PRINT X
```

Hinweis:

Entsprechend der anderen Adresse 48332 lautet die Zeile 20 beim C 64:

```
20 POKE 785,204:POKE 786,188
```

Nach RUN erhalten wir das Resultat 14, wie das Gesetz für INT es befiehlt. Natürlich hätten wir gleich PRINT INT (14.35) schreiben können, aber ich wollte ja nur demonstrieren. Der eigentliche Wert des USR-Befehls kommt hauptsächlich bei selbstgeschriebenen Maschinenprogrammen zum Zuge.

Sie können zur Übung im obigen Programm statt INT auch COS verwenden, indem Sie auf die Adresse 57935 ($E261) beziehungsweise beim C 64 auf 57938 ($E264) springen. Der Vergleich mit dem Basic-Befehl COS muß dasselbe Resultat ergeben.

Wer hat gemerkt, daß wir überhaupt nichts mit der Speicherzelle 0 gemacht haben, obwohl sie doch beim USR angeblich beteiligt ist?

Sie ist es wirklich, doch ohne unser Zutun. In diese Adresse wird beim Einschalten des Computers die Zahl 76 ($4C) geschrieben. Das ist der Code für den Maschinenbefehl »JMP«, der soviel bedeutet wie GOTO. Bei USR springt nämlich das Programm auf die Speicherzelle 0, findet dort den Sprungbefehl und in den nachfolgenden Zellen 1 und 2 die Sprungadresse – und führt den Sprung auch gleich aus.

und Anwendungen der PEEK- und POKEbaren Adressen, – sozusagen eine Wanderkarte mit Tourenvorschlägen und Sehenswürdigkeiten – in Form von Beispielen und Kochrezepten näherzubringen. Mir ist durchaus bewußt, daß das kein leichtes Unterfangen ist, da ich möglichst ohne Fach-Jargon auch für Nichttechniker verständlich bleiben möchte, und da die Zahl der zu behandelnden Adressen recht hoch ist. Ich werde also um Kompromisse wohl manchmal nicht herumkommen. Bevor wir anfangen, möchte ich noch einen kleinen »Arbeitsplan« machen.

■ Zur Methode:
Meine Erklärungen sind so aufgebaut, daß sie am besten vor dem Computer mit der Zeitschrift auf den Knien nachvollziehbar sind, also »Lies und Tipp«.

Längere Erklärungen oder Beispiele, die den Rahmen einer Adressenbeschreibung sprengen würden, oder die von generellem Interesse sind, werden in gesonderten Texteinschüben behandelt.

■ Zum Adressenbereich:
Prinzipiell sind natürlich alle RAM-Adressen (RAM = Lese- und Schreibspeicher) POKEbar und kämen daher in Betracht. Wir werden uns aber nur den Bereich von 0 bis 1023 vornehmen.

■ Zum Computer:
Der genannte Speicherbereich hat mit wenigen Ausnahmen für VC 20 und C 64 die gleiche Bedeutung. Ich werde daher beide Computer gleichzeitig behandeln und auf Unterschiede jeweils gezielt hinweisen.

■ Der erste Hinweis:
In Tabelle 1 sind die Unterschiede in groben Umrissen zusammengefaßt.

■ Zur Darstellung:
Die Kenntnis der Bedeutung dieser Speicherzellen kommt auch Programmen in Maschinensprache zugute. Ich gebe daher alle Adressen sowohl als Dezimal- als auch als Hexadezimalzahl (mit vorgestelltem »$«) an.

■ Zu den Adressen:
Wenn in die zur Diskussion stehenden Speicherzellen eine Adresse aus dem erlaubten Bereich 0 bis 65535 ($0 bis $FFFF) hineingeschrieben wird, geschieht das immer mit der Aufteilung in einen niederwertigen Teil (Low Byte) und einen höherwertigen Teil (High Byte). Das Rezept zur Umrechnung finden Sie im Texteinschub Nr. 2 »Die Low-/High-Byte-Darstellung«.

Adressen	Unterschied
0 – 2	sind verschieden
3 – 672	haben gleiche Funktionen
673 – 677	im VC 20 nicht benutzt
678 – 767	in beiden nicht benutzt
768 – 783	sind bei beiden gleich
784 – 787	im VC 20 nicht benutzt
788 – 819	haben gleiche Funktionen
820 – 1023	sind bei beiden gleich

Tabelle 1. Unterschiede zwischen VC 20 und C 64

Wozu brauchen das Betriebssystem und der Basic-Übersetzer RAM-Speicherzellen?

Auf den ersten Blick ist nicht verständlich, warum die Speicherzellen von 0 bis 1023 feste Bedeutung haben und für normale Programme nicht zur Verfügung stehen. Wenn sie schon, wie es heißt, vom Betriebssystem und dem Übersetzer-Programm verwendet werden, warum stehen sie dann nicht gleich im ROM-Speicher bei allen anderen Teilen dieser Systeme?

Ein Computer führt einen Programmschritt nach dem anderen aus, ganz stur, ohne eigene Entscheidungsfähigkeit, es sei denn, das Programm schreibt derartige Entscheidungen vor. Das Betriebssystem ist sozusagen im ROM eingefroren beziehungsweise festgeschrieben. Das würde aber bedeuten, daß der Computer keine Variationsmöglichkeiten hat, und daß alle Programme in gleicher Weise ablaufen. Aber das stimmt natürlich nicht! Alle Programme sind verschieden, sie belegen einen verschieden langen Speicherbereich und verarbeiten die unterschiedlichsten Variablen. Wir geben verschiedene Zeichen mit der Tastatur ein, der Computer wartet, bis eine Taste der Datasette gedrückt ist und so weiter.

Dafür braucht das Betriebssystem einen Speicherbereich, der variabel ist, in den es Zwischenwerte ablegen und später wieder auslesen kann.

Und das ist genau der Speicherbereich, der uns interessiert, nämlich von 0 bis 1023, womit wir wieder beim Thema wären.

Jetzt aber geht es los und zwar gleich in die vollen. Denn ausgerechnet die ersten drei Speicherzellen haben laut Tabelle bei beiden Computern eine verschiedene Bedeutung und zusätzlich gehören sie mit zu den kompliziertesten.

Adresse 0 bis 2 ($0 bis $2) beim VC 20:

Sprungbefehl und wählbare »Sprungadresse« des USR-Befehls.

Die drei Adressen werden bei der Abwicklung des Basic-Befehls USR verwendet.

Hinweis: Diesen drei Adressen des VC 20 entsprechen beim C 64 die Adressen 784 ($310) bis 786 ($312). Die folgenden Erklärungen gelten also entsprechend auch für den C 64. Mit dem Basic-Befehl USR wird ein Programm, das in Maschinensprache geschrieben ist, gestartet.

Wie das im einzelnen geht und welche Rolle dabei die Speicherzellen 0 bis 2 spielen, ist im Texteinschub Nr. 1 »Der USR-Befehl« näher beschrieben.

Adresse 0 ($0) beim C 64:

Datenrichtungsregister für Ein-/Ausgabe-Port des 6510-Mikroprozessors

Adresse 1 ($1) beim C 64:

Datenregister für Ein-/Ausgabe-Port des 6510-Mikroprozessors

Adresse 2 ($2) beim C 64:

unbenutzt

Im Gegensatz zum Mikroprozessor des VC 20 hat der des C 64 sechs Ein-/Ausgabe-Leitungen, die einzeln programmierbar sind und so eine direkte Verbindung zwischen dem Mikroprozessor und der Außenwelt herstellen. Warum nur sechs Leitungen und nicht wie üblich acht? Auf dem Chip selbst könnten acht Bit verkraftet werden, aber es stehen nur sechs Anschlußbeine zur Verfügung.

Um trotzdem flexibel zu bleiben, ist dieses Tor zum Prozessor – zutreffend auch »Port« genannt – in beiden Richtungen begehbar. Jede einzelne der sechs Leitungen kann vom Programmierer auf »Eingang« oder auf »Ausgang« geschaltet werden. Dazu dient das Datenrichtungsregister in der Speicherzelle 0.

Datenrichtungsregister in Zelle 0

Wenn zum Beispiel in das Bit 4 der Zelle 0 eine 0 hineingePOKEt wird, ist die Leitung Nummer 4 des Ports auf »Eingang« geschaltet. Es gilt für alle 6 Bit (Nummer 0 bis 5):
- Bit auf 0 = Eingang
- Bit auf 1 = Ausgang

Beim Einschalten schreibt das Betriebssystem in dieses Register die Dualzahl ..101111 (dezimal=47). Das heißt also, daß nur die Leitung Nummer 4 als Eingang verwendet wird, alle anderen aber als Ausgang. Warum das so ist, sehen wir gleich. Vorher will ich aber noch erwähnen, daß im C 64 von dieser Flexibilität des Mikroprozessor-Ports kein Gebrauch gemacht wird. Ich habe das ganze Betriebssystem durchgesehen, aber das einzige Mal, wo die Speicherzelle 0 angesprochen wird, ist eben bei der Einschaltroutine.

Das heißt aber nicht, daß Sie, lieber Hobby-Programmierer, darauf verzichten müssen. Ich kann mir vorstellen, daß besonders Ausgefuchste unter Ihnen durch POKEn eines anderen Bitmusters in die Speicherzelle 0 vielseitige Befehle erzeugen und einsetzen können.

Das wird besonders deutlich, wenn Sie jetzt sehen, mit welchen Teilen des Computers diese sechs Leitungen verbunden sind.

Datenregister in Speicherzelle 1

Mit diesem Register steuert der Mikroprozessor (und damit natürlich das Betriebssystem) die Auswahl von Speicherblöcken und den Betrieb mit dem Kassettenrecorder. Dem Programmierer steht diese Möglichkeit über POKEn auch zur Verfügung.

Bit 0
schaltet den Speicherbereich 40960 bis 49151 ($A000 bis $BFFF) zwischen dem Basic-Übersetzer (Interpreter) im ROM und freiem RAM um (Normalzustand=1).

Bit 1
schaltet den Speicherbereich 57344 bis 65535 ($E000 bis $FFFF) zwischen dem Betriebssystem (Kernel) im ROM und freiem RAM um (Normalzustand =1).

Bit 2
schaltet den Speicherbereich 53248 bis 57343 ($D000 bis $DFFF) zwischen Zeichen-ROM und Ein-/Ausgabe-ROM um (Normalzustand = 1).

Bit 3
sendet serielle Daten zum Kassettenrecorder (Normalzustand =0).
Bit 4
prüft, ob eine der Tasten des Recorders gedrückt ist, welche den Motor einschalten (Normalzustand =1).
Bit 5
schaltet den Motor des Recorders ein und aus (Normalzustand = 1).

Die RAM-ROM-Umschaltung

Sie wissen, daß Ihr C 64 deswegen so heißt, weil er 64 KByte Speicherplätze hat. Nur stimmt das nicht! Er hat nämlich 88 KByte und müßte eigentlich C 88 heißen.

Da mit den 16 Bit der High-/Low-Byte-Methode (siehe Texteinschub Nr. 2) nur 64 KByte adressierbar sind, müssen die restlichen 22KByte bei Bedarf eingeschoben werden – und das machen die oben erwähnten Bit 0 bis 2 des Datenregisters.

In Bild 1 sehen Sie die drei oben erwähnten Speicherblöcke, die sowohl mit RAM als auch mit ROM belegt sind, einer davon gleich doppelt. Ich habe ihnen folgende Namen gegeben:
- 40960 bis 49151 ($A000 bis $BFFF) = BLOCK A
- 53248 bis 57343 ($D000 bis $DFFF) = BLOCK D
- 57344 bis 65535 ($E000 bis $FFFF) = BLOCK E

Tabelle 2 gibt Ihnen die Übersicht über die gemeinsame Wirkung der Bit 0, 1 und 2 des Datenregisters auf den jeweiligen Inhalt der Speicherblöcke.

Der Vollständigkeit halber muß ich hier noch erwähnen, daß neben den drei ersten Bits der Speicherzelle 1 noch zwei weitere Signale die RAM/ROM-Umschaltung beeinflussen. Es sind das die Leitungen auf Pin 8 und 9 des Erweiterungssteckers (GAME und EXROM), welche durch Spiel- und Programmodule benutzt werden. Eine genaue Beschreibung der dadurch erzeugten sinnvollen Speicherkombinationen finden Sie in dem Buch »64 Intern« von Data Becker ab Seite 14. Zwei Anwendungsbeispiele dieser Umschaltung finden Sie im Texteinschub Nr. 3 »Manipuliertes Basic«.

Betrieb des Kassettenrecorders

Bit 3, 4 und 5 regeln, wie schon gesagt, den Betrieb des Kassettenrecorders.

Zu **Bit 3** ist oben schon alles Notwendige gesagt.

Bit 4 ist im Normalzustand auf 1, »normal« heißt hier, solange keine der Motor-Tasten der Datasette (PLAY, REWIND, FAST FORWARD) gedrückt ist. Zur Probe:

```
10 X=PEEK(1)
20 PRINT X
40 GOTO 10
```

Bit#	2	1	0	DEZ	HEX	BLOCK A	BLOCK D	BLOCK E
	1	1	1	55	$37	Basic	I/0	Kernal
	1	1	0	54	$36	RAM	I/0	Kernal
	1	0	1	53	$35	RAM	I/0	RAM
	1	0	0	52	$34	RAM	RAM	RAM
	0	1	1	51	$33	Basic	Zeichen	Kernal
	0	1	0	50	$32	RAM	Zeichen	Kernal
	0	0	1	49	$31	RAM	Zeichen	RAM
	0	0	0	48	$30	RAM	RAM	RAM

Dabei bedeuten
- Basic: Basic-Übersetzer (Interpreter)
- I/0: Ein-/Ausgabe-Register
- Zeichen: Zeichenspeicher
- Kernal: Betriebssystem
- RAM: frei verfügbarer Speicher

Tabelle 2. So sind die Bits 0, 1 und 2 des Datenregisters mit dem Inhalt der Blöcke A, D und E verknüpft.

Die schon erwähnte »Normalzahl« 55 (dual = 110111) läuft als Zahlenband solange, bis eine der besagten Tasten gedrückt wird. Dann läuft eine 7 (dual = 000111). Warum auch Bit 5 zu 0 wird, kommt später zur Sprache.

Mit einer kleinen Erweiterung der drei Zeilen können Sie in einem Programm den Status der Motor-Tasten abfragen. Ergänzen Sie:

```
30 IF X=7 THEN 50
50 PRINT »TASTE GEDRUECKT«
```

Um nur Bit 5 abzufragen, schreiben wir besser:

```
30 IF (X AND 16)=0 THEN 50
```

Diese Abfrage kann allerdings nicht unterscheiden, welche der drei Tasten der Datasette gedrückt worden ist. Außerdem funktioniert das alles nur, wenn – wie im »Normalfall« – das Bit 4 des Datenrichtungsregisters (Speicherzelle 0) auf 0 (Eingang) steht.

Bit 5 schaltet den Motor der Datasette ein und aus. Es bietet sich an, damit per Programm die Datasette zu schalten – wenn so etwas nützlich ist. Leider ist dieses Bit etwas schwieriger zu handhaben, da es in der Interrupt-Routine des Betriebssystems eine Rolle spielt.

Die Tasten der Datasette werden nämlich 60mal in der Sekunde abgefragt. Wenn keine Taste gedrückt ist, setzt das Betriebssystem sowohl das sogenannte »Interlock«-Register in Speicherzelle 192 auf 0 als auch Bit 5 der Zelle 1 auf 1, wodurch der Motor ausgeschaltet wird beziehungsweise

Texteinschub Nr. 2
Die Low-/High-Byte-Darstellung

Eine Speicherzelle der kleinen Commodore-Computer VC 20 und C 64 hat eine Länge von 8 Bit = 1 Byte. Mit diesen 8 Bit können Zahlen von 0 bis 255 ($00FF) dargestellt werden. Zur Darstellung von Zahlen über 255 verwenden wir die Low-/High-Byte-Methode.

Wir hängen einfach zwei Speicherzellen zusammen, mit deren 16 Bit wir Zahlen bis maximal 65535 ($FFFF) darstellen können. Die maximale Zahl 65535 ist übrigens auch die höchste Adresse des gesamten Speichers – was natürlich kein Zufall ist.

Ich will Ihnen jetzt zeigen, wie eine Dezimalzahl auf zwei 8-Bit-Speicherzellen verteilt wird, und umgekehrt, wie aus 2 Byte eine Dezimalzahl gebildet wird.

Schauen Sie sich das folgende Beispiel an:

DEZIMAL	47491			
DUALZAHL	1011	1001	1000	0011
HEX $	B	9	8	3
HIGH-BYTE	185		–	
LOW-BYTE	–		131	

Wir gehen von der Dezimalzahl 47491 aus. Ihre duale Darstellung mit 16 Bit – 1011100110000011 – teilen wir einfach in der Mitte und erhalten damit zwei neue Dual-Zahlen mit je 8 Bit = 1 Byte. Das linke Byte nennen wir »High-Byte«, da es den höheren Teil der Gesamtzahl darstellt. Das rechte Byte heißt entsprechend »Low-Byte«.

Jedes der beiden Bytes kann für sich allein in einer Speicherzelle untergebracht werden, in der natürlich dann der dezimale Wert des Bytes steht.

In der Tabelle habe ich zur Vollständigkeit noch die hexadezimalen Werte eingefügt, die sehr schön zeigen, daß der Vorteil dieser Zahlendarstellung darin liegt, daß jede Einzelziffer der 4-Bit-Dualzahl entspricht, genau so wie jede Zweiergruppe dem Byte (sowohl in Dual-, als auch in Dezimaldarstellung) und die vierstellige Zahl der großen Dezimal- und Dualzahl entspricht.

Zur Umrechnung der Low-/High-Bytes empfehle ich folgende Kochrezepte:

Dezimal in Low-/High-Byte

47491:256=185(High-Byte), Rest 131 (Low-Byte)

Der Rest fällt bei der Division per Hand automatisch an. Mit dem (Taschen-)Rechner erhält man den Rest durch:

185*256-47491=-131

Low-/High-Byte in Dezimal

High-Byte * 256 + Low Byte = Dezimal
185 * 256 + 131 = 47491

Wichtige Regel: Die Mikroprozessoren von VC 20 und C64 verlangen, daß immer das Low-Byte vor dem High-Byte kommen muß. Die Zahl wird sozusagen von rechts nach links gelesen (131/185).

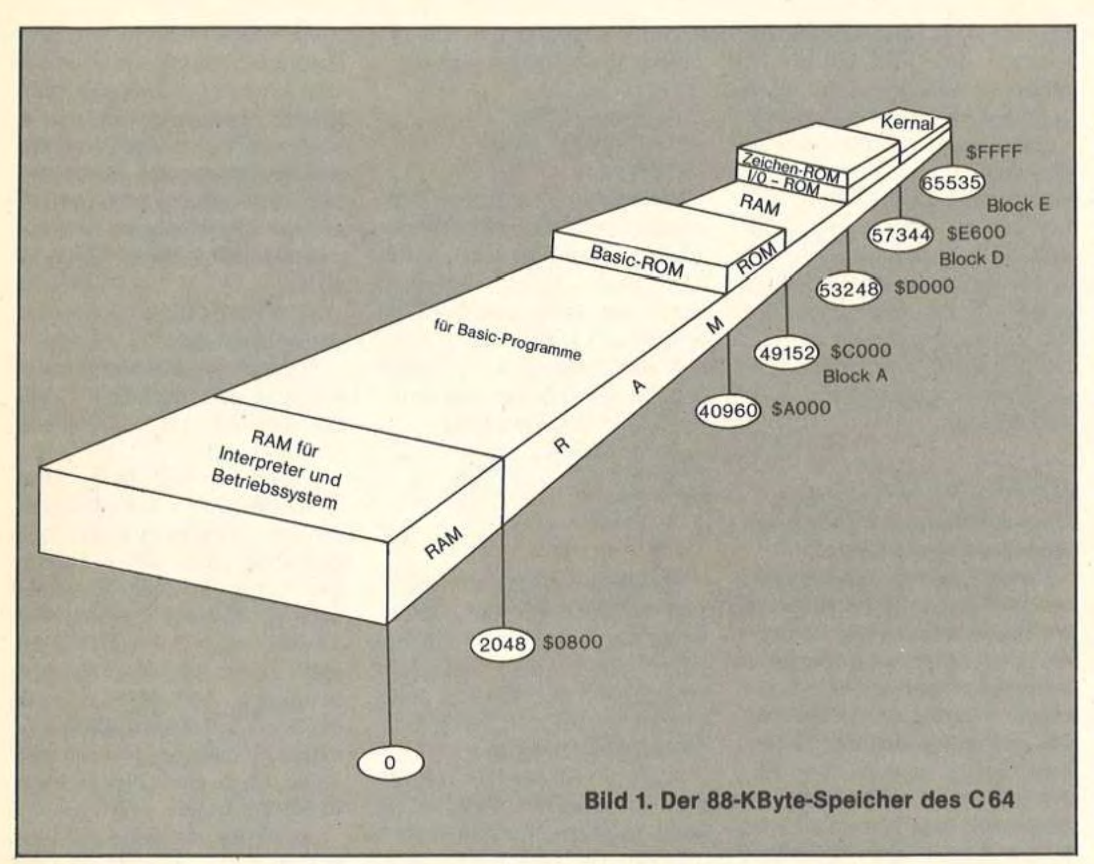


Bild 1. Der 88-KByte-Speicher des C64

bleibt. Da kann man nicht dagegen an. Wir haben nur eine Chance, wenn eine Taste bereits gedrückt ist und der Kassettenmotor schon läuft.

Dann nämlich können wir zuerst das Interlock-Register mit einem Wert größer als 0 lahmlegen:

```
POKE 192,255
```

Jetzt läßt sich der Motor der Datasette mit Bit 5 steuern:

```
POKE 1,39
```

beziehungsweise

```
POKE 1,PEEK(1) OR 32
```

schaltet den Motor aus,

```
POKE 1,7
```

beziehungsweise

```
POKE 1,PEEK(1) AND 31
```

schaltet den Motor ein.

Das Interlock-Register in Speicherzelle 192 werde ich später noch einmal erwähnen, da es seine Funktion auch beim VC 20 ausübt, allerdings mit anderen Ein-/Ausgangs-Ports. Das ist alles, was zur Speicherzelle 1 zu sagen ist.

Ab Speicherzelle 3 bis zur Speicherzelle 672 gelten alle

Texteinschub Nr. 3 Manipuliertes Basic

Wie Sie durch PRINT PEEK (1) selbst leicht feststellen, steht nach dem Einschalten des Computers im Register 1 die Zahl 55. In dualer Darstellung ist das 110111. Das entspricht dem in der ersten Zeile der Tabelle 2 dargestellten »Normalzustand« der einzelnen Bits.

Vergleichen Sie es bitte mit der Auflistung am Anfang der Beschreibung der Speicherzelle 1. Die in Tabelle 2 dargestellten Bits sind also die rechten drei Bits der Zelle 1.

Lassen wir die Bits 3, 4 und 5 unverändert, ergeben die acht Kombinationen der Tabelle 2 die Zahlen 55 bis 48. Durch den Befehl POKE 1,54 können wir nun den Basic-Übersetzer ausschalten und 8 KByte Speicher gewinnen. Nur nutzt uns das nicht viel, denn was tun – ohne Basic! Es gibt aber doch eine Anwendung. Zuvor will ich Ihnen aber noch beweisen, daß wir tatsächlich den Block A auf RAM umschalten. Der Trick besteht darin, den Basic-Übersetzer vom ROM in den darunter liegenden RAM umzuladen. Wenn er tatsächlich in RAM steht, müßten wir ihn durch POKEn verändern können zu einem Privat-Basic. Geben Sie direkt ein:

```
FOR J=40960 TO 49151: POKE J,
PEEK(J):
NEXT J
```

POKE J, PEEK (J) – das sieht dümmer aus als es ist. Die »Doppeldecker-Speicher« erlauben nämlich ein PEEKen nur aus dem ROM-Bereich. Ein hineinPOKEn dagegen geht nur in den RAM-Teil. Von dort aber kann er – wie gerade gesagt – nicht herausgelesen werden, es sei denn, wir schalten um!

Merken Sie was? Die Zeile oben liest also den Inhalt des Basic-ROMs und schreibt ihn in den RAM mit identischen Adressen. Die Ausführung der Zeile braucht einige Zeit. Wenn der Cursor wieder blinkt, schalten wir das RAM ein mit:

```
POKE 1,54
```

Wir merken natürlich noch keinen Unterschied, denn das RAM-Basic ist ja noch dasselbe.

Doch nun werden wir es verändern. In der Speicherzelle 41220 steht das »P« für den Befehl PRINT mit dem ASCII-Codewert 80. Dieses P ersetzen wir durch ein »G« (ASCII-Code = 71).

```
POKE 41220,71
```

Versuchen Sie bitte, mit dem (nicht durch »?« abgekürzten) PRINT-Befehl ein Zeichen auf den Bildschirm zu drucken. Es wird Ihnen nicht gelingen, denn der Befehl heißt jetzt:

```
GRINT "A"
```

was beweist, daß das Basic jetzt im RAM steht. Das Umdefinieren von Befehlen ist natürlich wenig sinnvoll. Aber wer die Maschinenprogramme des Basic kennt, kann sie auf diese Weise ändern, erweitern, einschränken, solange er sich auf in sich geschlossene Teile beschränkt.

Eine inzwischen oft zitierte Anwendung stammt von Jim Butterfield, den es begreiflicherweise stört, daß der Befehl ASC, welcher den ASCII-Code eines Strings erzeugt, bei einem Null-String das Programm mit ILLEGAL QUANTITY ERROR beendet. Versuchen Sie es:

`PRINT ASC ("A")` ergibt die Zahl 65.

`PRINT ASC ("")` hat die obige Fehlermeldung zur Folge.

Wenn Basic im RAM steht, können wir das ändern:

```
POKE 46991,5
```

Die Wiederholung des Befehls PRINT ASC ("") ergibt jetzt 0 – und, was das Wichtige ist, das Programm läuft weiter.

Durch zusätzliches Umladen des Speicherblocks E und anschließendes Umschalten mit POKE 1,53 ist auch das Betriebssystem veränderbar – ein weites Feld für fortgeschrittene Programmierer in Maschinensprache.

Die wohl wichtigste Anwendung der Umschaltmethode wird den Maschinen-Programmierern geboten, die dadurch eine kostenlose Speichererweiterung von 16 KByte erhalten. Bei gleichzeitiger Verwendung von Basic und Maschinenprogramm kann die Umschaltung besonders vorteilhaft eingesetzt werden. Das Umschaltprogramm muß dann aber ebenfalls in Maschinensprache geschrieben sein und darf nicht im Umschaltbereich liegen.

Das Umschalten von den Ein-/Ausgabe-Registern des Blocks D mit POKE 1,51 erlaubt, die Bitmuster der fest programmierten Zeichen aus dem Zeichen-ROM auszulesen, in einen freien RAM-Bereich zu bringen und dort dann nach eigenen Vorstellungen zu verändern.

Angaben sowohl für den C 64 als auch für den VC 20, zumindest was die Bedeutung der Zellen betrifft. Ihr Inhalt kann entsprechend der verschiedenen Adressen der Betriebssysteme voneinander abweichen. Wie üblich werde ich natürlich jeweils darauf aufmerksam machen.

Adresse 3 und 4 ($3 und $4)

Vektor auf die Routine zur Umwandlung einer Gleitkommazahl in eine ganze Zahl mit Vorzeichen

In diesen beiden Speicherzellen steht also ein Vektor. Was das ist, wird im Texteinschub Nr. 4 näher erläutert. Beim VC 20 deutet dieser Vektor auf die Adresse 53674 ($D1AA), beim C 64 auf 45482 ($B1AA). Sie können das mit

```
PRINT PEEK (3)+256*PEEK (4)
```

leicht nachprüfen. Ab diesen Adressen beginnt im Basic-Übersetzer (Interpreter) ein Programm, welches – natürlich in Maschinensprache – eine Gleitkommazahl in eine ganze Zahl umwandelt.

Diejenigen Leser, die mit Gleitkommazahlen nicht so vertraut sind, möchte ich auf den Texteinschub Nr. 5 verweisen. Er ist nur eine kleine Einführung. Eine detaillierte Beschreibung finden Sie im Assemblerkurs (Teil 8) von Heimo Ponnath (Ausgabe 4/85) beziehungsweise im 64er-Sonderheft 8/85, ab Seite 42.

Dieses Umwandlungsprogramm steht nicht nur den Maschinen, sondern auch den Basic-Programmierern zur Verfügung, allerdings nur über den USR-Befehl und da auch nur, wenn der »Floating Point Accumulator« #1 (FAC1) in den besagten Adressen 97 bis 102 mitbenutzt wird. Ich verschiebe daher alle weiteren Details auf unsere Ankunft bei diesen Speicherzellen.

Adresse 5 und 6 ($5 und $6)

Vektor auf die Routine zur Umwandlung einer ganzen Zahl in eine Gleitkommazahl

Dieses Programm ist die Umkehrung der oberen Routine. Sie beginnt beim VC 20 ab Speicherzelle 54161 ($D391), beim C 64 ab 45969 ($B391). Da hier prinzipiell dasselbe gilt wie oben, möchte ich nur kurz den Vorteil beleuchten, den derartige Vektoren haben. Eigentlich könnten wir direkt auf die im Vektor enthaltenen Adressen springen – wenn wir sie kennen.

Ein Sprung auf die Adresse des Vektors erlaubt uns jedoch immer die völlige Ignoranz seines Inhalts – und Commodore erlaubt die Änderung der Adressen im Basic-Übersetzer, wie es ja beim C 64 gegenüber dem VC 20 auch gemacht worden ist, ohne daß vorhandene Programme umgeschrieben werden müssen.

Adresse 7 ($7)

Suchzeichen zur Prüfung von Texteingaben in Basic

Diese Speicherzelle wird viel von denjenigen Basic-Routinen als Zwischenspeicher benutzt, die den direkt eingegebenen Text absuchen, um Steuerzeichen (Gänsefüße, Kommata, Doppelpunkte und die Zeilenbeendigung durch die RETURN-Taste) rechtzeitig zu erkennen. Normalerweise wird in der Zelle 7 der ASCII-Wert dieser Zeichen abgelegt. Die Speicherzelle 7 wird aber auch von anderen Basic-Routinen benutzt. Sie ist daher für den Programmierer praktisch nicht zu verwerten.

Adresse 8 ($8)

Suchzeichen speziell für Befehlsende und Gänsefüße

Wie Speicherzelle 7 dient auch die Zelle 8 als Zwischenspeicher für Basic-Texteingabe und zwar während der Umwandlung von Basic-Befehlen in den vom Computer verwendeten Befehlscode (Tokens). Die Speicherzelle 8 ist in Basic nicht verwertbar.

Adresse 9 ($9)

Spaltenposition des Cursors vor dem letzten TAB- oder SPC-Befehl

Speicherzelle 9 wird von den Basic-Befehlen TAB und SPC verwendet. Vor ihrer Ausführung wird die Nummer der Spalte, in der sich der Cursor befindet, aus der Speicherzelle 211 ($D3) nach 9 gebracht, von wo sie geholt wird, um die Position des Cursors nach der Ausführung von TAB und SPC auszurechnen.

Diese komplizierte Erklärung können wir durch Ausprobieren deutlicher machen. Dazu PRINTen wir 16mal den Buchstaben X hintereinander (Semikolon !), allerdings mit SPC (2) jeweils um 2 Spalten versetzt.

```
10 FOR I=0 TO 15
20 PRINT SPC (2) "X";
30 PRINT PEEK (9);
40 NEXT I
```

Nach jedem X wird durch Zeile 30 die »alte« Cursor-Spaltenposition ausgedruckt und zwar in derselben Zeile, ausgelöst durch das Semikolon. Dadurch erhöht sich laufend die in Speicherzelle 9 stehende Positionsangabe des Cursors. Wir erhalten folgenden Ausdruck:

```
..X.0...X.6...X.12...X
.19...X.26...X.33...X.
40...X.47...X.54...X.6
1...X.68...X.75...X.82
...X.1...X.7...X.13...
```

Sie können die Positionsnummer nachrechnen. Berücksichtigen Sie aber dabei, daß bei PRINT vor und nach jeder Zahl eine Stelle frei bleibt, die erste für das Vorzeichen, die zweite wegen des Abstandes.

Wichtig ist außerdem, daß die maximal mögliche Spaltenzahl nicht die Bildschirmspaltenzahl, sondern die »logische« Spaltenzahl ist, also 88 beim VC 20 und 80 beim C 64.

Wir können die Cursorposition in Adresse 9 auch abfragen und ein Programm damit steuern. Fügen Sie einfach in das obige Programm die folgende Zeile 35 ein:

```
35 IF PEEK (9)=33 THEN PRINT
"END": END
```

Sobald Position 33 erreicht ist, bleibt das Programm stehen.

Adresse 10 ($A)

Flagge für LOAD oder VERIFY

In Zelle 10 steht eine 0, wenn geladen wird und eine 1 bei einem VERIFY. Warum das so ist, will ich kurz erläutern:

Die Basic-Routinen für LOAD beziehungsweise für VERIFY sind völlig identisch. Was das Betriebssystem hinterher daraus machen muß, ist natürlich unterschiedlich. Das Basic erspart sich eine doppelte Routine, zeigt aber mit der Flagge in Speicherzelle 10 den Unterschied an.

Erwähnenswert ist noch, daß das Betriebssystem in einer Art Nationalismus seine eigene Flagge aufzieht: Den Unterschied zwischen LOAD und VERIFY speichert es seinerseits in Zelle 147 ($93) ab. Soweit ich es sehen kann, sind Inhalt und Bedeutung beider Speicherzellen völlig identisch.

Ich habe für Sie zwar kein Kochrezept zur Anwendung der LOAD-VERIFY-Flagge in einem Programm vorrätig, möchte Sie aber trotzdem ein bißchen zum Spielen anregen. Um meine Erklärung nachzuvollziehen, tippen Sie bitte direkt LOAD ein. Den Ladevorgang brechen Sie mit der STOP-Taste ab und fragen dann den Inhalt der Zelle 10 ab mit

```
PRINT PEEK (10)
```

Wir erhalten eine 0.

Wiederholen Sie bitte diesen Vorgang, aber mit VERIFY. Wir erhalten jetzt eine 1 – Quod erat demonstrandum.

Wir können auch in die Zelle 10 hineinPOKEn. Die »Wachablösung« zwischen Basic und Betriebssystem unter Hissen der Flagge in Zelle 10 findet beim VC 20 in der Speicherzelle 57705, beim C 64 in 57708 statt. Bevor wir diese Maschinenroutine mit SYS 57705 (SYS 57708) starten, geben wir mit dem Inhalt der Speicherzelle 10 an, ob es ein LOAD oder ein VERIFY sein soll.

Legen Sie ein Band mit Programm in die Datasette. Um ein LOAD zu erzeugen, geben wir direkt ein:

```
POKE 10,0:SYS 57705
(POKE 10,0:SYS 57708)
```

Entsprechend der Anweisung auf dem Bildschirm drücken Sie PLAY, und das Auffinden des ersten Programms wird mit LOAD gemeldet. Machen Sie das Ganze noch einmal, diesmal aber POKEn Sie bitte eine 1 in die Zelle 10. Jetzt meldet das Betriebssystem das Auffinden des Programms mit VERIFY.

Wie gesagt, vielleicht fällt Ihnen eine Anwendung dafür ein.

Adresse 11 ($B)

Flagge für den Eingabepuffer/Anzahl der Dimensionen von Zahlenfeldern (Arrays)

Alle Buchstaben und Zeichen, die mit der Tastatur direkt eingetippt werden, kommen in einen Eingabe-Pufferspeicher. Er beginnt ab Speicherzelle 512 ($200). Sobald die RETURN-Taste gedrückt wird, wandelt eine Routine des Basic-Übersetzers den Text in Codezahlen (Tokens) um. Diese Routine und eine andere, welche die Zeilen eines Programms aneinanderhängt, verwenden die Zelle 11 als Zwischenspeicher.

Sobald die Textumwandlung beendet ist, steht in Zelle 11 eine Zahl, die die Länge der Token-Zeile angibt.

Die Zelle 11 wird außerdem

noch von den Basic-Routinen benutzt, die ein Feld (Array) aufbauen oder ein bestimmtes Element in einem Array suchen. Was ein Feld oder Array ist, finden Sie in den Commodore-Handbüchern gut beschrieben. Außerdem gehe ich bei der Behandlung der Speicherzellen 47 bis 50 näher darauf ein.

Diese Routinen also verwenden die Speicherzelle 11, um die Anzahl der verlangten DIMensionen und den für ein neu aufgebautes Feld nötigen Speicherbedarf zu berechnen.

Adresse 12 ($C)

Flagge für Basic-Routinen, die ein Feld (Array) suchen beziehungsweise aufbauen

Diese Speicherzelle wird von den Basic-Routinen als Zwischenspeicher benutzt, die feststellen, ob eine Variable ein Feld (Array) ist, ob das Feld bereits DIMensioniert worden ist, oder ob ein neues Feld die unDIMensionierte Zahl von 11 Elementen hat.

Adresse 13 ($D)

Flagge zur Bestimmung des Datentyps (Zeichenkette/String oder Zahl)

Diese Flagge zeigt den Routinen des Basic-Übersetzers an, ob es sich bei den zur Verarbeitung anstehenden Daten um einen String oder um Zahlenwerte handelt. Zeigt die Flagge 255 ($FF), ist es ein String. Bei 0 handelt es sich um Zahlen. Diese Bestimmung erfolgt jedesmal, wenn eine Variable definiert oder gesucht wird. Diese Flagge kann leider nicht durch ein Basic-Programm abgefragt werden.

Adresse 14 ($E)

Flagge zur Bestimmung des Zahlentyps (Ganze Zahl oder Gleitkommazahl)

Sobald durch die Flagge in der vorherigen Zelle 13 eine Zahl signalisiert wird, steht hier die Zahl 128 ($80), wenn es sich um eine ganze Zahl handelt, während eine 0 die Zahl als Gleitkommazahl identifiziert.

Damit wollen wir ein bißchen experimentieren. Zeile 10 definiert eine Gleitkommazahl, Zeile 20 druckt sie und die Flagge aus Zelle 14 aus.

```
10 A=13.41
20 PRINT A,PEEK (14)
```

Wir erhalten die Zahl 13.41 und als Flagge eine 0.

```
30 B=INT (A)
40 PRINT B,PEEK (14)
```

INT bildet die ganze Zahl von 13.41. Also müßte die Flagge in Zelle 14 auf 128 stehen. Weit gefehlt! Da intern auch die 13 als Gleitkommazahl berechnet wird, erhalten wir immer noch eine 0.

```
50 B%=A
60 PRINT B%,PEEK (14)
```

Erst die Definition der Variablen B als ganze Zahl (mit %) ergibt die Flagge 128.

```
70 D=16*B%
80 PRINT D,PEEK (14)
```

Die Multiplikation einer ganzen Zahl mit der Ganzzahl-Variablen B% fällt in dieselbe Kategorie wie Zeile 30 oben, da die Verarbeitung als Gleitkommazahl erfolgt. Also erhalten wir zu Recht eine 0. Erst wenn D als ganze Zahl (Zeile 90) ausgewiesen wird, steht die Flagge wieder auf 128:

```
90 D%=16*B%
100 PRINT D%, PEEK (14)
```

Adresse 15 ($F)

Flagge bei LIST, Garbage Collection und Textumwandlung

Die Routine des LIST-Befehls muß unterscheiden zwischen Basic-Befehlen und normalem Text. Wenn eine Zeichenkette durch ein »Gänsefüßchen« identifiziert worden ist, wird die Flagge gesetzt, und der Text wird ausgedruckt.

Unter »Garbage Collection« (Müllabfuhr) wird die Routine des Betriebssystems verstanden, welche zu bestimmten Anlässen im Variablenspeicher alle nicht mehr benötigten Strings entfernt, um Platz zu schaffen. Dabei wird eine Flagge in Zelle 15 gesetzt, die anzeigt, daß eine Müllabfuhr bereits stattgefunden hat. Wenn bei der Speicherung eines neuen Strings zu wenig Speicherplatz vorhanden ist, wird bei der Flagge nachgesehen, ob gerade vorher schon durch die Müllabfuhr (Garbage Collection) der Speicher entrümpelt worden ist. Falls das der Fall ist, wird OUT OF MEMORY angezeigt, falls nicht, wird eine Müllabfuhr durchgeführt.

Schließlich wird Zelle 15 auch bei der Umwandlung von Basic-Befehlen in internen Codezahlen (Tokens) eingesetzt.

Adresse 16 ($10)

Flagge zur Anzeige eines Variablen-Feldes oder einer selbstdefinierten Funktion

Im Basic-Übersetzer gibt es eine Routine, die den Speicher absucht, ob es eine Variable mit bestimmten Namen bereits gibt. Wenn diese mit einer Klammer beginnt, wird die Flagge in Zelle 16 gesetzt, um anzuzeigen, daß es sich um eine Array-Variable oder um eine mit DEF FN selbstdefinierte Funktion handelt.

Adresse 17 ($11)

Flagge für INPUT, GET oder READ

Die Basic-Routinen für INPUT, GET und READ sind zum großen Teil identisch. Um Speicherplatz zu sparen, verwendet der Basic-Übersetzer die identischen Teile nur einmal. Um in die nichtidentischen Teile verzweigen zu können, wird in Zelle 17 angezeigt, um welchen der drei Befehle es sich gerade handelt. Die Flagge steht auf 0 für INPUT, auf 64 ($40) für GET und auf 152 ($98) für READ.

Mit dem folgenden kleinen Programm können wir das leicht nachprüfen.

Texteinschub Nr. 4
Zeiger, Vektoren und Flaggen

Zeiger und Vektoren sind Zahlenwerte, die jeweils in zwei benachbarten Speicherzellen stehen und in der Low-/High-Byte-Darstellung eine Adresse bilden.

Wir sprechen von einem »Zeiger«, wenn diese Adresse den Beginn von gespeicherten Daten angibt.

Ein »Vektor« zeigt ebenfalls auf eine Anfangsadresse, allerdings auf die eines Maschinenprogramms. Diese Unterscheidung wird leider nicht immer ganz eindeutig angewendet.

Eine »Flagge« besteht aus einem Zahlenwert in einer Speicherzelle, die von einem Programm dort abgelegt wird, um sich das Resultat einer Operation zu merken, beziehungsweise um es für eine spätere Verwendung bereitzuhalten.

Texteinschub Nr. 5
Die Zahlendarstellung bei den Commodore-Computern

Gleitkomma-Zahlen

Für diejenigen Leser, die das Thema der Zahlendarstellung in den Commodore-Handbüchern großzügig übersprungen haben, stelle ich es hier noch einmal vor.

Sie kennen die gängigen vier Zahlentypen:

- ganze Zahlen: 15, 21, 244
- Brüche; $\frac{2}{3}$, $\frac{26}{8}$, $\frac{15}{14}$
- negative Zahlen: -15, -255
- positive Zahlen: 10, 5, 123

Ganze Zahlen bereiten uns und dem Computer keine Probleme.

Bei Brüchen sieht es schon anders aus. Erinnern Sie sich an die Bruchrechnungsstunden in der Schule? Wieviel ist $\frac{51}{52}$ + $\frac{3}{4}$!!!

Ohne lange zu überlegen, rechnen wir natürlich um, $\frac{51}{52}$ = 0,9807692 und $\frac{3}{4}$ = 0,75; addiert ist das Resultat 1,7307692 – und schon sind Sie mitten in den Gleitkomma-Zahlen.

Bei obigem Beispiel gleitet allerdings noch nichts. Bei sehr großen oder aber auch sehr kleinen Bruch-Zahlen reicht uns – und einem Computer – nicht der Platz, um sie darzustellen. Die Zahl 0,000000000000000123 sprengt jeden normalen Rahmen.

Daher schreiben wir sie anders. Wir lassen das Komma nach rechts gleiten, bis es die erste Ziffer, die von 0 verschieden ist, findet und für jede Null, die es passiert, multiplizieren wir die Zahl mit 10.

Die Zahl oben sieht dann so aus:

0,123 x 10 hoch 15 (eine 1 mit 15 Nullen).

Die Grundzahl vorn heißt »Mantisse«, die 10 mit Hochzahl heißt »Exponent«.

Alle Commodore-Computer verarbeiten intern alle Zahlen in dieser Darstellung, also als Gleitkommazahl.

```
10 DATA 3
20 READ A
30 PRINT PEEK (17)
40 INPUT B
50 PRINT PEEK (17)
60 GET C$:IF C$= ""THEN 60
70 PRINT PEEK (17)
```

Zeile 10 und 20, 40 sowie 60 sind Anwendungen der drei zur Debatte stehenden Basic-Befehle. Nach der Durchführung jedes Befehls wird in den Zeilen 30, 50 und 70 die jeweilige Flagge ausgelesen.

Nach RUN erhalten wir als Resultat der Zeile 20 die Zahl 152, als Resultat von Zeile 30 die INPUT-Aufforderung mit Fragezeichen. Geben Sie irgendeine Zahl und RETURN ein. Wir erhalten so die 0. Die GET-Schleife in Zeile 40 wartet auf einen Tastendruck, dann erhalten wir 64.

Adresse 18 ($12)

1. Flagge für Vorzeichen bei SIN, COS und TAN
2. Flagge bei Vergleich

Zuerst kommt das Vorzeichen der trigonometrischen Funktionen an die Reihe.

Die Routinen des Basic-Übersetzers (Interpreter), welche die drei trigonometrischen Funktionen SIN, COS und TAN berechnen, verwenden die Speicherzelle 18 zur Bestimmung des Vorzeichens.

Zur Erinnerung: Die trigonometrischen Funktionen haben in den vier »Quadranten« des Kreises (0-90, 90-180, 180-270, 270-360 Grad) nicht unbedingt dieselben Vorzeichen. Die Vorzeichen ändern sich allerdings nur an den Grenzen der Quadranten, wie in Bild 2 zu sehen ist. Die Flagge in Zelle 18 gibt das Vorzeichen nicht direkt an, sondern auf Umwegen. Die Darstellung ist in der folgenden Tabelle zusammengefaßt.

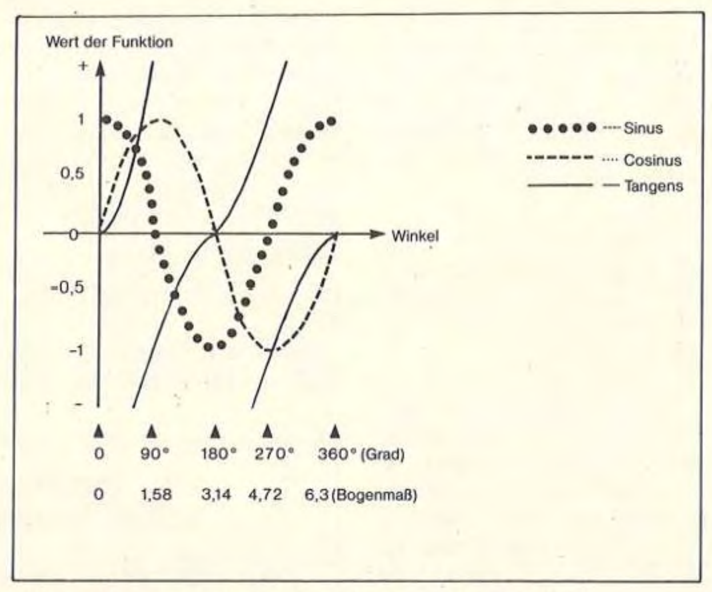


Bild 2. Trigonometrische Funktionen SIN, COS, TAN

und den Wert der Funktion mit Vorzeichen ausdruckt.

```
10 FOR I=0 TO 10 STEP 0.01
20 PRINT PEEK(18);INT
(I*100)/100;SIN(I):NEXT
```

Diese etwas umständliche Art, den Wert von I auszudrucken, vermeidet Rundungsfehler und begrenzt den Ausdruck auf zwei Dezimalstellen. Wenn Sie die Winkelwerte von I in Graden ausgedruckt haben wollen, können Sie eine andere Zeile 20 verwenden, welche die Umrechnungsformel vom Bogenmaß in Grade verwendet:

Winkel in Grad = Winkel im Bogenmaß * 180/π

```
20 Print PEEK(18);INT
(I*180/π);SIN(I):NEXT
```

Statt SIN können Sie genauso gut COS und TAN einsetzen.

In Bild 2 sind nicht nur die Kurven und die Bereiche der Vorzeichen, sondern auch die Winkelbereiche sowohl im Bogenmaß als auch in Graden dargestellt.

Die Speicherzelle 18 wird auch noch von anderen Routinen des Basic-Interpreters beansprucht und zwar von allen, die einen Vergleich wie <, >, >= und so weiter durchführen. Entsprechend der Art des Vergleichs steht dann in der Zelle 18 eine Ziffer von 0 bis 6.

Das folgende Programm macht das deutlich.

```
10 A=2
20 FOR I=1 TO 3
30 IF I= A THEN PRINT I;
PEEK(18);"="
40 IF I<>A THEN PRINT I;
PEEK(18);"><"
50 IF I> A THEN PRINT I;
PEEK(18);">"
60 IF I< A THEN PRINT I;
PEEK(18);"<"
70 IF I>=A THEN PRINT I;
PEEK(18);">"="
80 IF I<=A THEN PRINT I;
PEEK(18);"<"="
90 IF I<A OR I=A THEN
PRINT I;PEEK(18);"< OR ="
100 NEXT I
```

WINKEL	0-90	90-180	180-270	270-360
SIN	gleich	Wechsel	Wechsel	gleich
COS	255	255	0	0
TAN	0	255	255	0

Dabei bedeutet »gleich«: 0-0-0-0 oder 255-255-255 »Wechsel«: 0-255-0-255

Da die Erklärung mit »gleich« beziehungsweise »Wechsel« nicht gerade einleuchtend ist, schlage ich vor, daß Sie sich das Ganze mit dem folgenden kleinen Programm selbst anschauen, welches für viele Werte des Winkels im Bogenmaß – und in kleinen Schritten – den Wert der Flagge, daneben den Winkel I

Kurz zur Erklärung dieser Zeilen: In der FOR..NEXT-Schleife wird die Variable I mit der Konstanten A=2 verglichen. In den Zeilen 30 bis 90 werden alle möglichen Vergleichsoperatoren durchgeprüft. Jeder der zutrifft, druckt den Wert von I, den Wert der dann in Zelle 18 stehenden Flagge und schließlich den Vergleichsoperator aus. Aus dem Resultat dieses Programms läßt sich folgende Tabelle zusammenstellen:

Vergleich	Flagge in 18
< OR =	0
> OR =	0
>	1
=	2
>=	3
<	4
<>	5
<=	6

Sie sehen, die Flagge für die kombinierten Vergleichsoperatoren entspricht der Summe ihrer Einzelwerte. Nur die Verknüpfung über OR nicht, denn die ergibt 0.

Adresse 19 ($13)

Flagge zur Kennzeichnung des laufenden Ein-/Ausgabegerätes

Immer dann, wenn von Basic Daten ein- oder ausgegeben werden, schaut die entsprechende Routine des Übersetzers in Zelle 19 nach, um welches Peripheriegerät es sich handelt. Zur Debatte stehen Tastatur, Datasette, RS232-User-Port, Bildschirm, Drucker und Floppy-Laufwerk.

Die Flagge ihrerseits ist ausschlaggebend für die feinen Unterschiede, wie zum Beispiel das Fragezeichen, bei Eingabe von der Tastatur (INPUT) oder die Anweisung »Press Play on Tape« bei Eingabe von der Datasette.

Beim Einschalten des Rechners setzt die Initialisierungsroutine des Betriebssystems, die beim VC 20 ab Adresse 58276 ($E3A4), beim C 64 ab 58303 ($E3BF) beginnt, die Flagge in Zelle 19 auf 0. Die Null bedeutet Eingabe über Tastatur und Ausgabe über Bildschirm.

Wenn Sie einen Disassembler haben, drucken Sie doch einmal das Assemblerlisting aus. Sie werden in Adresse 58324/58325 ($E3D4/E3D5), beim C 64 in 58354/58355 ($E3F2/E3F3) den Befehl finden, der eine Null nach Zelle 19 ($13) bringt.

Immer dann, wenn ein Programm nicht Tastatur und Bildschirm, sondern eines der oben genannten anderen Peripheriegeräte anspricht (indem mit OPEN.... eine Datei = Logical File eröffnet wird), wird in Zelle 19 die Nummer der gerade bearbeiteten Datei eingetragen, mit den bereits beschriebenen Konsequenzen.

Ich will hier nicht weiter darauf eingehen, da wir den Inhalt von Zelle 19 selbst nicht auslesen können. Er wird nämlich immer gleich wieder auf Null gesetzt.

Wir können ihn aber durch POKE verändern. Durch POKE 19,1 gaukeln wir dem Rechner vor, daß Ein- und Ausgabe über »externe« Geräte läuft, selbst wenn nur die Tastatur und der Bildschirm betrieben werden.

Wenn zum Beispiel der Rechner der Meinung ist, daß ein INPUT von der Datasette kommt, druckt er kein Fragezeichen aus; auch kein EXTRA IGNORED als Fehlermeldung bei zu zahlreicher Eingabe und das alleinige Drücken der RETURN-Taste ignoriert er auch, im Gegensatz zum »normalen« INPUT.

Probieren Sie es aus:

```
10 INPUT "TEST";A$
20 PRINT A$
```

In diesem Normalfall erscheint nach RUN darunter die Aufforderung TEST?

Eine Eingabe, zum Beispiel XX, erscheint mit einem Abstand daneben, und nach RETURN wird XX an den Anfang der nächsten Zeile gedruckt. Alle falschen Eingaben werden mit den üblichen Fehlermeldungen quittiert.

Jetzt fügen wir ein:

```
5 POKE 19,1
```

Nach RUN erscheint wieder die Aufforderung TEST, aber ohne Fragezeichen. Die Eingabe XX wird ohne Abstand daneben gesetzt und nach RETURN mit einem Abstand in derselben Zeile weitergeschrieben.

Das Drücken der RETURN-Taste setzt den Cursor nicht wie üblich in die nächste Zeile, sondern schiebt ihn in derselben Zeile weiter.

Diesen zusätzlichen Effekt muß man beachten, da er sehr störend für den Verlauf eines Programms sein kann.

Man kann ihn natürlich auch nutzbringend einsetzen, hat er doch die Eigenschaft eines automatischen »Cursor UP«. Eine pfiffige Anwendung dieser Art wurde von Brad Templeton für den PET erfunden und ist von Jim Butterfield für eine MERGE-Routine mit dem Namen »Magic Merge« veröffentlicht worden.

Da diese Routine aber primär auf der Eigenschaft der Speicherzelle 153 basiert, werde ich sie dann erläutern, sobald wir bei der Zelle 153 angelangt sind.

Zurück zur Flagge in Zelle 19. Umgekehrt können wir POKE 19,0 leider nicht nutzen, da die betroffenen Befehle GET, GET#, INPUT, INPUT# und PRINT# die Flagge sofort auf den richtigen Wert setzen. Nur PRINT und LIST tun das nicht, wie wir bei dem PRINT-Befehl oben ja gesehen haben.

Adresse 20 und 21 ($14 und $15)

Zeilennummer für LIST, GOTO, GOSUB und ON, Zeiger der Adresse bei PEEK, POKE, SYS und WAIT

In diesen Speicherzellen wird die Zeilennummer der Sprungbefehle GOTO, ON..GOTO und GOSUB sowie die Zeilenangabe beim LIST-Befehl gespeichert. Da die Werte bis maximal 65535 gehen können, braucht der Computer 2 Byte zur High-/Low-Byte-Darstellung.

Die GOTO-Routine (im VC 20 ab 51360 = $C8A0, im C 64 ab 43168 = $A8A0) vergleicht die Zahl in 20 und 21 mit der laufenden Zeilenzahl. Wenn sie kleiner ist, wird ab der ersten Zeile des Programms gesucht. Ist sie aber größer, dann beginnt die Suche ab der laufenden Zeilenzahl. Die Suche geht solange, bis die in 20 und 21 angegebene Zeilenzahl gefunden ist. Dann fährt das Programm mit dieser Zeile fort.

LIST speichert in 20 und 21 die höchste auszulistende Zeilennummer ab, falls keine Angabe beim LISTen gegeben worden ist, den Wert 65535 ($FFFF).

Die Befehle PEEK, POKE, SYS und WAIT verwenden diese Speicherzellen zur Angabe der Adressen, die dem Befehl immer folgen müssen.

Leider können wir die Speicherzellen 20 und 21 mit Basic-Programmen nicht bearbeiten; ihr Inhalt wird immer gleich auf 20 zurückgesetzt.

Adresse 22 ($16)

Zeiger auf den nächsten freien Speicherplatz im »Temporary String Descriptor Stack«

Dieser Zeiger bezieht sich in seiner Wirkung auf die übernächsten Speicherzellen 25 bis 33 ($19 bis $21).

Diese werden als Stapelspeicher (Stack) für Angaben über vorläufige Zeichenketten – auf englisch »Temporary String Descriptor« – verwendet.

Die Speicherzelle 22 ($16) ihrerseits enthält einen Zeiger auf den jeweils nächsten verfügbaren Platz in diesem Speicher ab Zelle 25. Da er eine Kapazität von 3 * 3 Byte hat, zeigt der Zeiger auf die Zelle 25 ($19), wenn er leer ist. Bei einem Eintrag zeigt er auf 28 ($1C), bei zwei Einträgen auf 31 ($1F) und schließlich auf 34 ($22), wenn der Speicher voll ist.

Eine Zeichenkette ist dann »vorläufig«, wenn sie noch nicht einer Stringvariablen zugeordnet worden ist, zum Beispiel »Mahlzeit« in dem Basic-Befehl

```
PRINT "MAHLZEIT".
```

Beim Einschalten setzt das Betriebssystem mit der Einschaltroutine ab Adresse 58303 ($E3BF) im C 64, beim VC 20 ab 58276 ($E3A4) den Zeiger auf 25. Die Stringverwaltungsroutine ab 46215 ($B487) im C 64 beziehungsweise ab 54407 ($D487) im VC 20 fragt bei String-Eingaben die Flagge ab. Nach jeder Eintragung in den Speicher ab Zelle 25 wird der Zeiger um 3 weitergesetzt.

Sie können die Leerflagge 25 mit

```
PRINT PEEK (22)
```

leicht nachprüfen.

Die anderen Eintragungen können nicht nachgeprüft werden, weil sie sofort auf 25 zurückgesetzt werden.

Wir können sie aber durch POKE beeinflussen; ob das sinnvoll ist, ist eine andere Frage.

```
10 POKE 22,34
20 PRINT "MAHLZEIT"
```

Die Zahl 34 in Zelle 22 sagt dem Programm, daß der Speicher ab Zelle 25 voll ist. Wir bekommen statt der MAHLZEIT eine Fehlermeldung serviert.

Mit einem POKE-Befehl, der als Argument die für den vorgesehenen Zweck ungültige Zahl 35 verwendet:

```
POKE 22,35
```

erreichen wir allerdings zwei interessante »Dreckeffekte«. Zum einen unterdrückt der Befehl die Ausgabe des READY, zum anderen aber bewirkt er, daß der LIST ein Listing ohne Zeilennummern ausgedruckt wird, sowohl auf dem Bildschirm als auch mit dem Drucker.

Das billigste editierfähige Textverarbeitungssystem

Die Idee dazu habe ich von Mike Apsey's Hinweis in »Commodore User« Juli 1984. Mit Zeilennummern versehen, läßt sich jeder beliebige Text schreiben, verbessern, verschieben, abspeichern, aber nicht RUNen!!

Der POKE-Befehl von oben (POKE 22,35) gefolgt von einem CMD und LIST, druckt dann alles brav als reinen Text aus. Die maximale Zeilenlänge entspricht der Zeilenlänge des jeweiligen Computers.

Probieren Sie es aus:

```
10 DER COMPUTER BIETET
IN DER
20 DATENFERNÜBER-
TRAGUNG
30 UNGEAHNTE
MÖGLICHKEITEN.
40 ABER DIE GEFAHR
50 USW. USW.
60:
```

Jede Zeile wird mit der RETURN-Taste abgeschlossen. Damit auch alles gedruckt wird, muß – zumindest bei meinem Drucker (1526) – eine »Leerzeile« folgen (Zeile 60). Mit

```
POKE 22,35:OPEN 1,4:CMD
1:LIST
```

wird der Text ohne Zeilennummern ausgedruckt. Sie können ihn vorher nach Belieben verändern.

Wie gesagt, nur nicht mit RUN starten, denn das bringt unweigerlich eine Fehlermeldung.

Adresse 23 und 24 ($17 und $18)

Zeiger auf die Adresse der letzten Zeichenkette im »Temporary String Stack«

Der Inhalt dieser 2 Byte zeigt auf den zuletzt benutzten Speicherplatz innerhalb der Adresse 22 bis 33. Das heißt, daß der Wert in 23 ($17) immer um 3 kleiner ist als der in 22 ($16), während der Wert in 24 ($18) eine Null ist.

Adresse 25 bis 33 ($19 bis $21)

Stapelspeicher für Angaben über vorläufige Zeichenketten

Das ist also der Speicherbereich, von dem in den beiden vorigen Abschnitten dauernd die Rede war. Ich gebe zu, »Descriptor Stack for Temporary Strings« drückt die Sache präziser aus als der deutsche Text.

Die Bedeutung eines »vorläufigen« Strings habe ich oben in der Beschreibung der Speicherzelle 22 erklärt.

Was ein Stapelspeicher (Stack) ist, entnehmen Sie bitte dem Texteinschub 6. Jeder der 3 Byte langen Angaben im Stack von 22 bis 33 enthält die Länge sowie die Anfangs- und Endadressen eines vorläufigen Strings, ausgedruckt als Verschiebung im Basic-Speicherbereich.

Adresse 34 bis 37 ($22 bis $25)

Verschiedene Zwischenspeicher

Diese vier Speicherzellen werden vom Basic-Übersetzer (Interpreter) für verschiedene Zwischenergebnisse und Flaggen benutzt, die aber dem Programmierer nichts nutzen.

Adresse 38 bis 42 ($26 bis $2A)

Arbeitsspeicher für arithmetische Operationen

Diese Speicherzellen werden von den Basic-Routinen bei der

Multiplikation und Division als »Notizblatt« verwendet. Auch die Routinen, welche die erforderliche Speichergröße beim Definieren eines Zahlenfeldes (Array) ausrechnen, benutzen diesen Bereich.

Adresse 43 und 44 ($2B und $2C)

Zeiger auf den Anfang der Basic-Programme im Speicher

Dieser Zeiger, in der Low-/High-Byte-Darstellung, gibt dem Basic-Übersetzer an, ab welcher Speicherzelle das Basic-Programm beginnt. Normalerweise ist diese Adresse fest vorgegeben. Beim C 64 zum Beispiel zeigt der Zeiger auf 2049 ($801). Beim VC 20 ist die Lage schon schwieriger, denn der Speicherbeginn hängt davon ab, welche Speichererweiterung eingesetzt ist. Die folgende Tabelle 3 gibt darüber Auskunft.

Tabelle 3: Beginn des Programmspeichers

C 64	2049 ($801)
VC 20 (GV)	4097 ($1001)
VC 20 (+3 K)	1025 ($401)
VC 20 (+ 8 K)	4609 ($1201)

Mit dem Befehl

```
PRINT PEEK (43) + PEEK
(44)*256
```

läßt sich der jeweilige Beginn des Programmspeichers leicht feststellen. Mit einem POKE-Befehl kann der Programmierer diese Anfangsadresse verändern. Wozu das gut ist, fragen Sie?

Anwendung #1:

Nun, wenn Sie zum Beispiel ein Maschinenprogramm mit einem Basic-Programm gemeinsam betreiben wollen, brauchen Sie einen Speicherbereich für das Maschinenprogramm, der vom Basic-Programm nicht belegt wird. Wir sprechen vom »Schützen des Maschinenprogramms vor dem Überschreiben durch das Basic«. Der Speicherbereich eines Maschinenprogramms ist immer bekannt. Nach seinem letzten Speicherplatz kann das Basic-Programm beginnen.

Die Verschiebung der Anfangsadresse erfolgt in vier Schritten:

1. Schritt: In den Speicherplatz vor dem neuen Basic-Bereich muß eine Null gePOKEt werden. Die Null dient zum Abgrenzen.
2. Schritt: Die Adresse der ersten Speicherzelle wird in die Low-/High-Byte-Darstellung umgerechnet. Ich verweise dazu auf die Erklärung dieses Vorgangs im Texteinschub Nr. 1.
3. Schritt: Das Low-Byte wird in die Speicherzelle 43, das High-Byte in die Zelle 44 gePOKEt.
4. Schritt: Die Operation muß unbedingt mit dem Befehl NEW abgeschlossen werden, um sicherzustellen, daß auch alle anderen Zeiger auf ihren Anfangszustand gesetzt werden.

Im folgenden kleinen Programm wird angenommen, daß der Speicher bis zur Adresse 5000 ($1388) durch ein Maschinenprogramm belegt ist. Das Basic-Programm kann daher ab 5002 ($138A) anfangen, denn in 5001 muß ja eine Null stehen. Die Adresse 5002 teilt sich auf in ein High-Byte von INT (5002/256) = 19 und ein Low-Byte von 5002-(19*256) = 138.

```
10 POKE 5001,0
20 POKE 43,138
30 POKE 44,19
40 NEW
```

Der Effekt einer solchen »Verbiegung« des Zeigers in 43 und 44 wird im Texteinschub Nr. 7 »Der sichtbare Basic-Speicher« demonstriert.

Neben der oben erwähnten Anwendung der Zeigerverbiegung gibt es noch andere Möglichkeiten:

Anwendung #2:

Christoph Sauer hat in seinem Kurs »Der gläserne VC 20« in Ausgabe 10/84 auf Seite 158 gezeigt, wie man mehrere Programme gleichzeitig im Speicher unterbringen und zwischen ihnen umschalten kann.

Anwendung #3:

Man kann zwei oder mehrere unabhängige Programme genau hintereinander in den Speicher bringen, um sie aneinander zu hängen, was dem im Commodore-Basic fehlenden Befehl MERGE entspricht. Dabei dürfen die Zeilennummern sich allerdings nicht überschneiden.

Anwendung #4:

Durch Hinaufschieben des Basic-Bereichs kann Platz geschaffen werden für selbstdefinierte Zeichen oder hochauflösende Grafik.

Die Speicherzellen-Paare von 45, 46 bis 55, 56 ($37 bis $38) zeigen auf weitere für Basic-Programme wichtige Speicherbereiche, die deswegen gemeinsam betrachtet werden sollten. Bild 5 stellt den Zusammenhang grafisch dar. In diesem Bereich werden alle Variablen eines Programms gespeichert. Zur Erinnerung:

Texteinschub Nr. 6
Was ist ein Stapelspeicher (Stack)?

Der normale Arbeitsspeicher des Computers, auf englisch »Random Access Memory« oder kurz RAM genannt, hat für jede Speicherzelle eine eigene Adresse, die beim Schreiben in den Speicher oder beim Lesen aus dem Speicher angegeben werden muß.

Als Analogie möge eine Aktenablage dienen, bei der jeder Akt (Brief, Papier, Zeichnung) in einen Ordner kommt, mit Nummer versehen.

Um einen Akt herauszuholen, muß man die Nummer (Adresse) kennen, unter der er abgelegt ist.

Ein Stapelspeicher, auf englisch »Stack« genannt, funktioniert wie eine Aktenablage, bei der jeder Akt einfach oben auf einen Stapel gelegt wird, daher der Name. Diese Ablage erfolgt ohne Kennzeichnung oder Nummer, einfach immer der Reihe nach.

Einen Akt kann man aus einem Stapelspeicher nicht beliebig herausholen, da immer nur der oberste Akt zugänglich ist.

Die Methode der Stapelspeicher bietet sich überall dort an, wo es auf die Reihenfolge der gespeicherten Daten ankommt. Basic merkt sich zum Beispiel der Reihe nach die Adressen, von denen aus mit GOSUB ein Unterprogramm angesprungen wird. Wenn mehrere GOSUBs hintereinander eingesetzt werden, liegt auf dem Stapel immer die letzte Absprungadresse bereit zum Rücksprung.

Ein Stapelspeicher hat demnach nur eine einzige Adresse, die sowohl zum Abspeichern als auch zum Auslesen dieselbe ist.

Voraussetzung eines Stapelspeichers ist natürlich eine Routine, welche alle gespeicherten Daten im Stapelspeicher um einen Platz weiterschiebt, wenn eine neue Information »oben auf den Stapel gelegt wird«.

Das Basic der Commodore-Computer verwendet mehrere dieser Stapelspeicher.

Die Programmiersprache Forth ist völlig auf dem Prinzip des Stapelspeichers aufgebaut.

Texteinschub Nr. 7
Der sichtbare Basic-Speicher

Wenn wir den Variablen A die Adresse des Speicherbeginns der Basic-Programme zuordnen und dann mit einer FOR..NEXT-Schleife den Inhalt dieser und der nächsten 100 Speicherplätze ausdrucken, sehen wir in dezimaler Darstellung die ersten 101 Zahlenwerte, mit denen der Computer ein Basic-Programm speichert.

Ein Verbiegen des Zeigers in Speicherzelle 43/44 kann auf diese Weise in seiner Wirkung sichtbar gemacht werden.

Als Demo-Programm wähle ich zwei Zeilen, welche die Zahlen 1 bis 9 und die Buchstaben A bis I ausdrucken.

```
 10 PRINT "123456789"
 20 PRINT "ABCDEFGHI"
100 A=2049 : REM*C 64
      4097 : REM*VC 20 ohne Erweiterung
      1025 : REM*VC 20 mit 3 KByte
      4609 : REM*VC 20 mit 8 KByte oder mehr
110 PRINT CHR$(147)
```

Zeile 100 definiert den Speicheranfang. Zeile 110 löscht den Bildschirm.

```
120 FOR J=A TO A+100
130 PRINT PEEK (J);
140 NEXT J
```

Die Befehle in den Zeilen 120 bis 140 drucken den Inhalt der ersten 101 Zellen dieses Basic-Programms aus. SAVEn Sie bitte dieses kleine Programm, denn wir brauchen es noch einmal. Dann geht es los mit RUN. In Bild 3 ist der Bildschirm-Ausdruck des VC 20 mit 8 KByte dargestellt, der des C 64 zeigt praktisch dieselbe Information.

Überspringen Sie bitte zunächst die ersten beiden Zahlen. Die dritte und vierte Zahl ist 10 und 0. Das ist (als Low- und High-Byte) die Nummer der ersten Zeile des Basic-Programms. Dann folgt 153, das ist der interne Codewert für PRINT. Diese Codes für alle Basic-Befehlswörter heißen »TOKEN«, sie sind im Texteinschub Nr. 32 angegeben.

Die nächste Zahl auf dem Bildschirm ist die 34, sie ist der ASCII-Code für den Gänsefuß. Danach folgen in aufsteigender Reihenfolge die ASCII-Codes der Ziffern 1 (48) bis 9 (57). Danach sehen Sie wieder den Gänsefuß (34). Schließlich kommt eine Null als Abstandszeichen zur nächsten Basic-Zeile.

Machen Sie bitte folgendes Experiment: Ausgehend von der Adresse der ersten auf dem Bildschirm ausgedruckten Speicherzellen – zum Beispiel 4609 beim VC 20 mit 8 KByte – zählen Sie die Zellen weiter bis zur Abgrenzungs-Null. In meinem Beispiel steht die Null in Zeile 4625. Das heißt, daß die nächste Basic-Zeile in 4626 anfängt. Und das ist genau die Zahl, die in den ersten beiden Zellen steht, die wir vorhin übersprungen haben; in meinem Beispiel steht da 18 18. Machen wir die Probe: 18 + 256 * 18 = 4626.

Jede Basic-Zeile im Speicher beginnt also mit der Adresse der nächsten Zeile (sie heißt Koppeladresse) und endet mit einer Null.

Ab 4626 folgt dann die nächste Koppeladresse, danach mit 20 0 die Zeilennummer, und Sie erkennen jetzt sicher die Codes der Angaben von Zeile 20 wieder.

So, jetzt wollen wir den Zeiger in 43 und 44 verbiegen. Ich schlage vor, daß wir den Basic-Beginn um zehn Adressen höher schieben. Sie müssen jetzt die in Zeile 100 oben verwendete Zahl für A in die High-/Low-Byte-Darstellung umrechnen und das Low-Byte um 10 erhöhen. Dieses Zahlenpaar POKEn wir in die Zellen 43 und 44. Vorher müssen wir aber noch in Zelle (A + 10) –1 eine Abstands-Null POKEn.

Wir geben diese Befehlssequenz im Direktmodus ein:

☐ für den C 64:
```
POKE 2058,0:POKE 43,11:POKE 44,8:NEW
```
☐ für den VC 20 (GV):
```
POKE 5006,0:POKE 43,143:POKE 44,19:NEW
```
☐ für den VC 20 (= 3 KByte):
```
POKE 1034,0:POKE 43,11:POKE 44,4:NEW
```
☐ für den VC 20 (> 8 KByte)
```
POKE 4618,0:POKE 43,11:POKE 44,18: NEW
```

Jetzt ist der Anfang des Basic-Speichers versetzt. Um das zu prüfen, geben wir das kleine Programm von oben nochmal ein und lassen es mit RUN laufen. Der resultierende Bildschirmausdruck ist in Bild 4 dargestellt.

Die ersten Zahlen sind genauso wie vorher. Es sind auch die Reste von vorher, da wir den Speicher nicht auf Null gesetzt haben. Aber zählen Sie bitte die ersten zehn Adressen hoch. Da finden Sie unser Programm von vorhin genau wieder, beginnend mit der Abstands-Null. Aber Vorsicht, lassen Sie sich nicht verwirren, denn die Koppeladressen sind natürlich jetzt auch jeweils um 10 höher. Aber hinter den Koppeladressen finden wir wieder unser Programm, in gleicher Weise dargestellt wie beim ersten Mal. Da der Zeiger in 43 und 44 von allen entsprechenden Routinen des Übersetzers und des Betriebssystems abgefragt wird, läuft ein verschobenes Programm fehlerfrei, solange natürlich der Zeiger nicht wieder verändert wird.

Bild 3. Ausdruck vom VC 20 mit 8 KByte Speichererweiterung

Bild 4. Neuer Bildschirmausdruck

Wir unterscheiden zwischen »normalen« Variablen (numerische und String-Variable) und Feld-Variablen (Arrays). Dabei ist wichtig zu wissen, daß ein Basic-Programm während des Eintippens oder Einladens von Disk beziehungsweise Kassette in den 1. Block kommt. Während des Programmlaufs werden alle normalen Variablen in den 2. Block geschrieben, alle Felder (Arrays) in den 3. Block und schließlich der Text der Zeichenketten (Strings) sozusagen rückwärts vom Ende des Arbeitsspeichers in den 4. Block. Je nach Größe des Programms und nach Anzahl der Variablen wandern die Blockgrenzen nach oben beziehungsweise die von Block 4 nach unten. Wenn sie sich treffen beziehungsweise überschneiden, gibt es »OUT OF MEMORY«.

Diese Blockbewegung ist in Bild 5 durch die Pfeile dargestellt.

Adresse 45 und 46 ($2D und $2E)

Zeiger auf die Anfangsadresse des Speicherbereichs für Variable

Dieser Zeiger, in der Low/High-Byte-Darstellung, gibt dem Basic-Interpreter an, ab welcher Speicherzelle die Variablen eines Basic-Programms gespeichert sind. Da die Variablen direkt an das Basic-Programm anschließen, zeigt dieser Zeiger natürlich gleichzeitig auf das Ende des Basic-Programms.

Es muß betont werden, daß es sich nur um den Bereich der »normalen« Variablen handelt, also nicht um Felder (Arrays). Anders als der Zeiger in 43 und 44, der auf fest definierte Speicherzellen zeigt, liegt der Zeiger für den Variablen-Beginn nicht fest. Je nach Länge des Programms wandert er nach oben.

Sobald ein Programm eingetippt oder aus einem externen Speicher (Diskette, Kassette) eingelesen ist, wird der Zeiger in 45 und 46 durch RUN auf ein Byte hinter das Programmende gesetzt und alle Variablen werden in der Reihenfolge ihres Auftretens gespeichert. Da normalerweise die Länge eines Basic-Programms während des Ablaufs konstant bleibt, werden die Variablen in ihrer Position auch nicht gestört.

Das bedeutet, daß sie sowohl vom Programm als auch vom Programmierer nach einer Unterbrechung abgefragt werden können. Nur wenn das Programm modifiziert wird, wandert der Zeiger zusammen mit den Variablen entsprechend weiter.

Denselben Effekt wie das oben erwähnte RUN haben übrigens auch die Befehle NEW, CLR und LOAD. Eine Ausnahme bildet das LOAD innerhalb eines Programms, welches den Zeiger nicht zurücksetzt. Dadurch wird ein Aneinanderhängen von mehreren Programmen samt Variablen-Weiterverwendung unter bestimmten Voraussetzungen ermöglicht.

Die Bearbeitung der Variablen durch das Basic-Programm und die daraus resultierenden Kochrezepte für den Programmierer sind im Texteinschub Nr. 8 »Normale Variable in BASIC« separat erläutert.

Die verschiedenen Typen der Variablen und ihre Darstellung

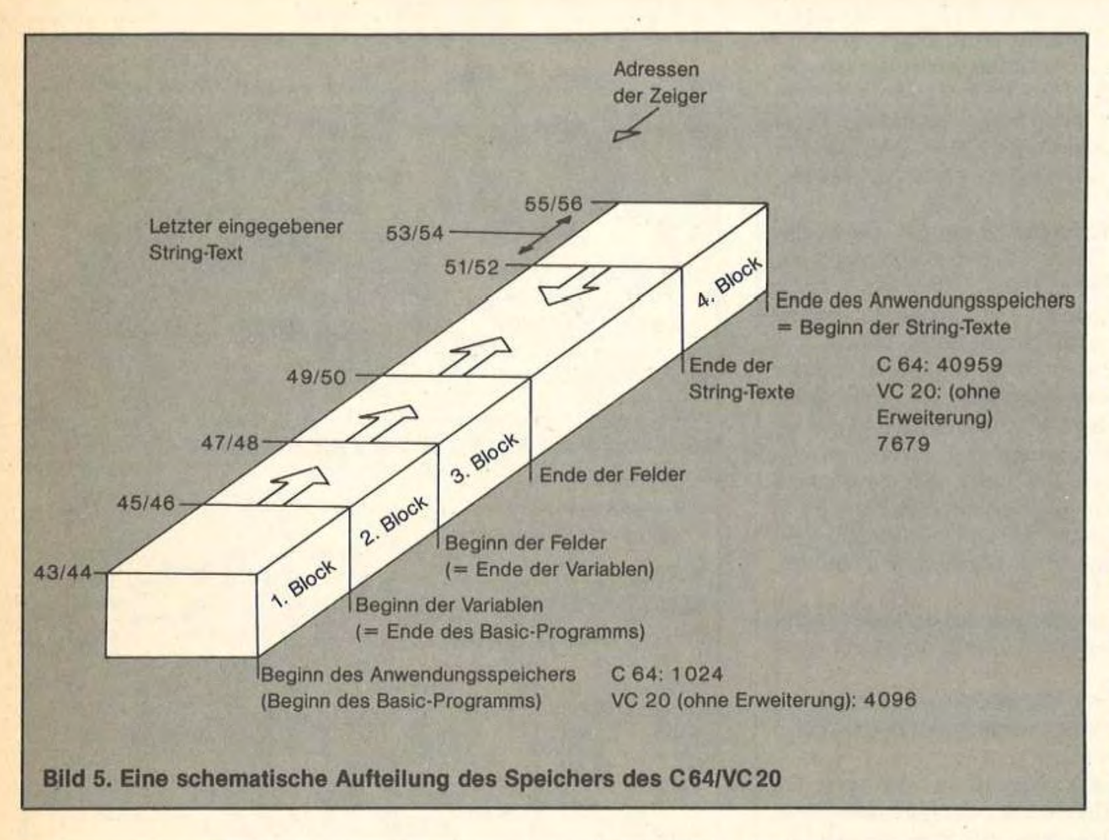


Bild 5. Eine schematische Aufteilung des Speichers des C64/VC20

im Speicher finden Sie im 64'er, Ausgabe 10/84, Seite 157 und noch ausführlicher in Ausgabe 11/84, Seite 124, dargestellt und erklärt.

Für diejenigen Leser, welche kein Monitor- beziehungsweise Disassembler-Programm haben oder benutzen können, ist im Texteinschub Nr. 9 »Darstellung der normalen Variablen im Speicher« eine kleine Anleitung gegeben, wie sie die Variablendarstellung mittels Basic anschauen können.

Adresse 47 und 48 ($2F und $30)

Zeiger auf die Anfangsadresse des Speicherbereichs für Felder (Arrays)

Dieser Zeiger, in der Low-/High-Byte-Darstellung, gibt dem Basic-Übersetzer (Interpreter) an, ab welcher Speicherzelle die Felder (Arrays) eines Basic-Programms gespeichert

Texteinschub Nr. 8 Normale Variable in Basic

Alle Daten, die in einem Basic-Programm nicht in Form von READ-DATA-Anweisungen vorkommen, werden als »Variable« unmittelbar nach dem Basic-Programm abgespeichert. Wir unterscheiden dabei zwei Typen:
- normale Variable
- Felder (Arrays)

Wir betrachten hier nur die »normalen« Variablen.

Sie erscheinen in dem Speicherbereich, dessen Beginn durch den Zeiger in den Zellen 45 und 46 und dessen Ende durch den Zeiger in 47 und 48 angegeben wird, in derselben Reihenfolge, in welcher sie während des Ablaufes des Basic-Programms auftreten. Wenn Basic dann auf eine der bereits definierten (und abgespeicherten) Variablen zurückgreifen soll, muß es den gesamten Variablenbereich von Anfang an absuchen, bis es den Namen der gesuchten Variablen gefunden hat. Wenn diese Variable ganz am Ende des Bereiches steht, kann dieser Suchprozeß recht lange dauern.

Regel 1:

Häufig vorkommende Variable sollen am Anfang des Variablenbereichs stehen. Das wird dadurch erreicht, daß sie als erste Variable in einem Programm »definiert« werden. Falls sie erst später im Programm gebraucht werden (aber dann häufig), werden sie trotzdem am Anfang des Programms angegeben, notfalls mit einem beliebigen Wert, der später dann keine Rolle mehr spielt und ersetzt wird. Man nennt das einen »Dummy«-Wert.

Die Felder-Variablen stehen direkt nach den »normalen« Variablen. Auch hier kann der gewiefte Programmierer Gutes tun. Wenn nämlich nach einer Definition eines Feldes später im Programm noch normale Variable dazukommen, ist natürlich zuerst kein Platz für sie da. Das Betriebssystem des Computers muß erst alle Felder-Variablen weiterschieben, bevor die Neuankömmlinge in dem dadurch erweiterten Variablenbereich gespeichert werden können. Auch das kostet unnötig viel Zeit.

Regel 2:

Alle normalen Variablen sollen als erste in einem Programm definiert werden. Wer also drauflos programmiert, sollte zumindest am Ende das Programm so umbauen, daß diese simple Regel erfüllt wird.

Texteinschub Nr. 9 Darstellung der normalen Variablen im Speicher

Die normalen Variablen kommen in drei Arten vor:
- ganzzahlige Variablen
- Gleitkomma-Variablen
- String-Variablen (Zeichenketten)

Der Unterschied zwischen den drei Typen ist in den Commodore-Handbüchern gut erklärt, und ich verzichte hier auf eine Wiederholung. Ich will vielmehr direkt zeigen, wie die Variablen im Speicher abgelegt sind.

Wir können den Speicher direkt sichtbar machen.

Einmal geht das in Maschinencode mittels eines Monitors beziehungsweise Disassemblers.

Zum anderen aber geht das auch in Basic und zwar mit folgendem Trick, den ich Th. und M.L. Beyer (MC 10/1983) abgeschaut habe.

Wir verlegen den Beginn des Basic-Variablenspeichers einfach auf den Beginn des Bildschirmspeichers. Auf diese Weise können wir zwar kein vernünftiges Programm laufen lassen, aber alle direkt eingegebenen Variablen-Definitionen werden sofort sichtbar, weil sie eben im Bildschirmspeicher stehen.

Wir erreichen die Verlegung des Speichers durch »Verbiegen« der Zeiger in den Zellen 45 und 46 und 47 und 48. Die Bedeutung dieser Zeiger ist ja im Kurs erklärt.

Die Speicherverlegung beziehungsweise die Methode dazu ist für den C 64 anders als für den VC 20.

■ **VC 20:**

Alle Angaben gelten für den VC 20 ohne Speichererweiterung, also ziehen Sie bitte alle Speichermodule heraus. Der Speicherbereich für Programme und deren Variablen beginnt jetzt ab Adresse 4096, das ist Block 1 im Bild 5. Der Bildschirmspeicher beginnt ab 7680. Wir verlegen jetzt den Bildschirmspeicher in den Block 1, so daß er ebenfalls ab Adresse 4096 beginnt. Danach müssen wir noch eine Farbe – am besten Schwarz – in den Farbspeicher POKEn, der in dieser neuen Konfiguration von 37888 bis 38399 liegt. Warum das so ist, erklärt Christoph Sauer in seinem Aufsatz »Der gläserne VC 20«, Teil 4, im 64'er 1/85, Seite 131.

Das High-Byte der Adresse, in welcher der Bildschirmspeicher

beginnt, steht in der Speicherzelle 648. Sie können das jederzeit mit PRINT PEEK(648) nachprüfen. Umgekehrt können wir eine Zahl hineinPOKEn, wodurch der Bildschirmspeicher verschoben wird. In unserem Fall erhalten wir das High-Byte für 4096 durch 4096/256 = 16.

Machen Sie jetzt bitte folgende Schritte:
1) direkt eingeben: `POKE 648,16(RETURN)`,
2) RUN/STOP und RESTORE drücken, bis der Cursor wieder da ist,
3) direkt eingeben:
`FOR J = 37888: TO 38399: POKE J,0: NEXT J (RETURN)`,
4) mit der DELETE-Taste (nicht mit CLR !) den ganzen Text des Bildschirms löschen,
5) mit dem Cursor etwa acht Zeilen nach unten gehen,
6) mit der Commodore- und SHIFT-Taste zusammen auf die Groß- und Kleinschrift umstellen.

Schritt 1 und 3 habe ich oben schon erklärt. Schritt 4 ist nicht absolut notwendig, aber ein leerer Bildschirm ist für uns besser. Die CLR-Taste würde Schritt 3 zunichte machen. Schritt 5 erlaubt uns, weiter unter auf dem Bildschirm Variablen einzugeben, ohne den oberen Teil vollzuschreiben. Schritt 6 schließlich erleichtert das Erkennen der Variablen-Darstelllung.

■ C 64:

Beim C 64 beginnt der Bildschirmspeicher ab 1024. In Low-/High-Byte-Darstellung ist das 0/4 (1024/256=4, Rest 0). Geben Sie bitte direkt ein:

```
POKE 46,4 :POKE 48,4
```

Das Low-Byte in 45 und 47 können wir weglassen, da es ja in beiden Fällen 0 ist. Diese Methode gilt für alle neueren C 64, bei denen direkt ein Zeichen in den Bildschirmspeicher gePOKEt werden kann, ohne sich um die Zeichenfarbe kümmern zu müssen. Es gibt noch einige C 64 mit älteren Betriebssystemen, bei denen die Zeichenfarbe auch angegeben werden muß. Hier gilt:

```
FOR J = OTO 1000:POKE 55296+3,14:NEXT J
```


■ Alles weitere gilt für beide Computertypen gleich

Wenn Sie jetzt den Bildschirm löschen, den Cursor ungefähr in die Mitte des Bildschirms fahren und wiederum direkt eingeben:
`VARIABLE = 3` und die RETURN-Taste drücken,
dann erscheinen oben sieben Zeichen. Bitte schalten Sie mit der SHIFT- und Commodore-Taste auf den zweiten Zeichensatz um, jetzt können wir besser lesen.

Aus anderen Kursen wissen Sie wahrscheinlich, daß Variable mit 7 Byte dargestellt werden. In der Tat sehen wir oben die ersten beiden Buchstaben des Variablennamens VA und fünf weitere Zeichen. Wir wollen aber systematisch vorgehen und uns zuerst die ganzzahligen Variablen anschauen.

Ganzzahl-Variable

Wiederholen Sie bitte den Vorgang (Löschen, Cursor auf Mitte, 2. Zeichensatz). Jetzt geben Sie eine Ganzzahl-Variable ein:
`VA%=3`
Nach RETURN sehen wir als erstes Zeichen ein reverses V, dann ein reverses A, den Klammeraffen @, das kleine c und nochmals drei @. Die beiden ersten Zeichen des Variablennamens (besteht er nur aus einem Zeichen, wird mit einer 0 aufgefüllt) werden mit ihrem ASCII-Code eingegeben, zu dem bei Ganzzahl-Variablen zur Kennzeichnung einer solchen die Zahl 128 addiert wird.

Schauen Sie in einer ASCII-Tabelle (64'er, Ausgabe 7/84) nach: Das V hat 86, um 128 erhöht gibt das 214. Wir arbeiten hier aber im Bildschirmspeicher, der die Zahlen auf seine eigene Weise interpretiert, nämlich als Bildschirmcode. Der Bildschirmcode-Tabelle entnehmen wir das Zeichen für den Wert 214 und das ist das invertierte V. Für das A können Sie das selbst nachvollziehen.

Also: In unserer Darstellung erkennen wir Ganzzahl-Variable an den invertierten Zeichen des Namens.

Das 3. und 4. Zeichen sind das High- und Low-Byte des Variablenwertes und zwar im Bildschirmcode. In unserem Beispiel der 3 ist das High-Byte 0, also der Klammeraffe @, das Low-Byte 3, also das c. Die restlichen drei Byte sind mit 0 aufgefüllt.

Wenn Sie mit dem Cursor auf die 3 fahren, es mit einer 5 überschreiben und RETURN drücken, verwandelt sich das c in ein e. Beim Überschreiben mit 255 erscheint als 4. Byte das Zeichen für den Bildschirmcode 255. Beim Überschreiben mit 257 ändern sich beide Bytes. Das 3. (High-)Byte springt auf a (=1), das 4. (Low-)Byte ebenfalls auf a. Nun, 1 * 256+1 = 257.

Während, wie bewiesen, das Low-Byte von 0 bis 255 gehen kann, sind beim High-Byte nur Werte zwischen 0 und 127 zugelassen. Die Werte ab 128 signalisieren negative Zahlen. Probieren Sie es aus:
`127 * 256+255=32767`
Ein Überschreiben mit 32767 resultiert in einer Darstellung der Zeichen für den Bildschirmcode 127 und 255. Der Wert 32768 wird nicht mehr akzeptiert. Dasselbe machen wir noch schnell für negative Zahlen.

Überschreiben Sie bitte die letzte Zahl mit 0. Wie zu erwarten war, sind Byte 4 und 5 jetzt 0 (Klammeraffe).

Wenn Sie jetzt mit -1 überschreiben, erscheint für beide Bytes das Zeichen mit dem Bildschirmcode 255. Bei -2 sehen wir die Zeichen mit den Code-Werten 255 und 254.

Sie sehen also, daß die negativen Zahlen für ganzzahlige Variable sozusagen vom Ende der Tabelle her dargestellt werden, wobei die höchste negative Zahl wieder 32767 ist. Diese »Rückwärtszählung« ist bedingt durch die Methode der negativen Zahlendarstellung im Zweierkomplement. Der Platz und die Gelegenheit verbieten es mir, näher darauf einzugehen. Aber ich glaube, unser kleines Experiment hat Ihnen zumindest von der Darstellung her den Zusammenhang gezeigt. In Bild 6 ist diese Darstellung der ganzzahligen Variablen im Speicher wiedergegeben.

1	2	3	4	5	6	7
Erstes	Zweites	High-	Low-			
Zeichen des Variablen-Namens (ASCII-Wert + 128)		Byte des Variablenwertes		0	0	0

Bild 6. So stehen ganzzahlige Variable im Speicher

Gleitkomma-Variable

Ich hoffe, Sie verzeihen mir, wenn ich diese Darstellung an dieser Stelle überspringe. Sie ist nämlich nicht ganz leicht zu verstehen, und ich möchte sie lieber dann im Detail erklären, wenn wir zur Diskussion der Speicherzellen 97 bis 101, nämlich des Gleitkomma-Akkumulators kommen. Da geht es in einem Stück. Als Vorgeschmack gebe ich jetzt in Bild 7 nur die Zusammenfassung an.

1	2	3	4	5	6	7
Erstes	Zweites	Exponent + 129	Mantisse mit Genauigkeit von 32 Dualstellen, 1. Bit des 1. Bytes ist das Vorzeichen			
Zeichen des Variablen-Namens (ASCII-Wert)						

Bild 7. Der FAC = Gleit- oder Fließkomma-Akkumulator

String-Variable

Zuerst ist es erforderlich, den Computer in den Anfangszustand zurückzusetzen. Wenn Sie einen RESET-Schalter haben, bitte diesen drücken, sonst aber aus- und einschalten. Wir geben nach Löschen des Bildschirms in der unteren Hälfte direkt ein:
`ZX$="A" <RETURN>`
Wir erhalten ein Z, ein invertiertes X, ein kleines a, ein grafisches Zeichen, eine Leerstelle und zwei Klammeraffen.

Fahren Sie bitte jetzt mit dem Cursor auf das A und ändern den String um in BC. Nach RETURN verwandelt sich das a in das b,

das 4. Zeichen ändert sich ebenfalls. Die ersten beiden Zeichen sind also wieder der Name der Variable.

Um zu kennzeichnen, daß es eine String-Variable ist, erscheint das 2. Zeichen des Namens invertiert. Wie oben entsteht es dadurch, daß zum ASCII-Code die Zahl 128 addiert wird. Diese Zahl wird aber wieder als Bildschirmcode interpretiert und entsprechend angezeigt (vergleichen Sie es mit den ASCII- und Bildschirmcode-Tabellen).

Das 3. Zeichen gibt die Länge des Strings an, also im ersten Fall mit a (=1 im Bildschirmcode), im 2. Fall mit b (=2). Zeichen 4 und 5 geben als Low- und High-Byte die Adresse an, bei der begonnen wird, den Text des Strings zu speichern. Das können wir nachprüfen.

Wir hatten die beiden Fälle:
1. ZX $ = "A"
4. Zeichen: (Bildschirmcode: 255) und 5. Zeichen: (Bildschirmcode 156) ergibt als Adresse 40959.
2. ZX$ = "BC"
4. Zeichen: (Bildschirmcode 253) und 5. Zeichen: (Bildschirmcode 156) ergibt als Adresse 40957.

Der Text der Zeichenketten wird am Ende des Arbeitsspeichers (40959 beim C 64, 7679 beim VC 20 ohne Erweiterung) abgelegt und zwar von hinten nach vorn.

Mit `PRINT PEEK(40957);PEEK(40958);PEEK(40959)` drucken wir den Inhalt dieser Speicherzellen aus und erhalten: 66 67 65. Im ASCII-Code ist das: B C A. Die Zusammenfassung für String-Variable (Bild 8) sieht so aus:

1	2	3	4	5	6	7
Erstes	Zweites	Anzahl der Zeichen des Strings	Low-	High-		
Zeichen des Variablen-Namens			Byte der Adresse, ab welcher der Text des Strings abgespeichert ist		0	0
ASCII-Wert	ASCII-Wert+128					

Bild 8. String-Variable

sind. Was Felder sind und wozu sie gebraucht werden, ist im Texteinschub Nr. 10 kurz erläutert. Da die Felder direkt nach den normalen Variablen gespeichert werden, zeigt dieser Zeiger natürlich gleichzeitig auf das Ende des Speichers für normale Variablen.

Durch POKEn einer Adresse in die Speicherzellen 47 und 48 kann der Speicherbereich am Anfang eines Programms beinahe beliebig verschoben werden. Beinahe deswegen, weil die Verschiebung im Zusammenhang mit den anderen Bereichen (siehe Bild 5) einen Sinn haben muß. Im übrigen gilt für diesen Zeiger dasselbe, was schon für den Zeiger in 45 und 46 gesagt worden ist. Die Darstellung der Feld-Variablen selbst kann mit der Methode angesehen werden, die im Texteinschub Nr. 11 erklärt ist.

Wie aus den Erklärungen hervorgeht, wird bei Feldern mit Zeichenketten (Strings) in dem von Zeiger 47 und 48 bezeichneten Speicherbereich nur die Definition beziehungsweise die Dimensionierung gespeichert. Die eigentlichen Zeichenketten stehen wie bei den normalen Variablen im vierten Block, vom Speicherende rückwärts angeordnet.

Adresse 49 und 50 ($31 und $32)

Zeiger auf die Endadresse (+1) des Speicherbereichs für Felder (Arrays)

Der Inhalt dieser Speicherzellen zeigt auf die Adresse, wo der Speicherbereich für Felder aufhört. Wie aus Bild 5 hervorgeht, werden die Zeichenketten vom Ende des verfügbaren RAM-Speichers rückwärts gespeichert. Man kann also auch sagen, daß der Zeiger in 49 und 50 die letzte mögliche Adresse für Zeichenketten angibt. Wenn in einem Programm neue Variablen definiert werden, rutscht diese Adresse weiter nach oben und nähert sich dem Ende der Zeichenketten, die durch den Zeiger in 51 und 52 angegeben wird.

Wenn sich die Speicherbereiche der Felder und Zeichenketten berühren, bleibt der Computer stehen und führt die »Garbage Collection« (Müllabfuhr) durch – ein Prozeß, in dem nicht mehr gebrauchte Zeichenketten entfernt und der Zeichenketten-Speicher reduziert wird. Ist danach immer noch kein Platz, wird OUT OF MEMORY gegeben.

Der Befehl FRE löst immer eine solche Garbage Collection aus und gibt dann die Differenz zwischen den Adressen in den Zeigern 49 und 50 und 51 und 52 als verbleibenden, noch verfügbaren, Speicherbereich aus.

Adresse 51 und 52 ($33 und $34)

Zeiger auf die untere Grenze des Speicherbereichs für den Text der Zeichenketten-Variablen

Der Inhalt dieser Speicherzellen zeigt in Low-/High-Byte-Darstellung auf das jeweilige untere Ende (siehe Bild 5) des Textspeichers von Zeichenketten. Er bezeichnet aber zugleich auch das obere Ende des frei verfügbaren RAM-Bereichs. Das entsteht dadurch, daß der Text der Zeichenketten vom Ende des RAM-Bereichs nach unten gespeichert wird. In Bild 5 ist das durch den Pfeil dargestellt.

Beim Einschalten des Computers und nach einem RESET wird dieser Zeiger auf das oberste Ende des RAM-Bereichs gesetzt. Beim C 64 ist das 40960 ($A000). Beim VC 20 hängt es von den eingesetzten Speichererweiterungen ab, ohne Erweiterung ist die Adresse 7680 ($1E00).

Der Befehl CLR setzt den Zeiger auf die Adresse, welche durch den Zeiger in den Speicherzellen 55 und 56 als das Ende des Basic-Speichers angegeben wird. Wozu das dient, erkläre ich Ihnen bei der Beschreibung dieses Zeigers weiter unten.

Adresse 53 und 54 ($35 und $36)

Zeiger auf die Adresse der zuletzt eingegebenen Zeichenkette

In diesen Speicherplätzen steht die Adresse (im vierten Block, siehe Bild 5) der Zeichenkette, die als letzte von Routinen (Programme, Direkteingabe) zur String-Manipulation abgespeichert worden ist. Mit dem folgenden kleinen Programm können Sie das genau sehen:

```
10 PRINT PEEK(53)+
256*PEEK(54),
20 PRINT PEEK(51)+256*PEEK
(52)
30 INPUT A$
40 GOTO 10
```

Zeile 10 druckt uns zuerst (links) den Zeiger auf die zuletzt eingegebene Zeichenkette aus, Zeile 20 rechts daneben den Zeiger auf die untere Speichergrenze der Zeichenketten. Zeile 30 fordert zur Eingabe einer Zeichenkette auf.

Wenn Sie bei frisch eingeschaltetem Computer das Programm starten, sehen Sie eine 0 (=vorher noch kein String eingegeben) und daneben die Adresse dezimal 40960 (C 64) beziehungsweise dezimal 7680 (VC 20 ohne Erweiterung). Wenn Sie auf das Fragezeichen des INPUT hin zum Beispiel ein A eintippen, erhalten Sie links den vorigen Wert von rechts und rechts jetzt eine um 1 kleinere Zahl. Eine weitere Eingabe von zum Beispiel XXXXX schiebt die alte rechte Zahl nach links und die neue wird um die Anzahl der Zeichen, also 5, verringert.

Adresse 55 und 56 ($37 und $38)

Zeiger auf das Ende des für Basic-Programme verfügbaren Speichers

Dieser Zeiger, in der Low-/High-Byte-Darstellung, gibt dem Basic-Übersetzer an, welches die höchste von Basic verwendbare Speicheradresse ist. Wie aus Bild 5 ersichtlich, ist diese Adresse zugleich der Anfang der als Variable abgespeicherten Zeichenkette (Strings).

Normalerweise ist diese Adresse fest vorgegeben. Die folgende Tabelle 4 gibt darüber Auskunft:

Tabelle 4. Ende des Programmspeichers

	Adresse	Zeiger in 55 56
C 64	40960	0160
VC 20 (Grundv.)	7680	030
VC 20 (+3 KByte)	7680	030
VC 20 (+8 KByte)	16384	064

VC 20 (+16 KByte)	24576	096
VC 20 (+24 KByte)	32768	0128

Beim Einschalten des Computers überprüft das Betriebssystem den gesamten RAM-Speicher, bis es zur ersten ROM-Speicherzelle kommt, setzt den Zeiger in 55 und 56 auf diese Adresse und druckt den bekannten Kopf mit der verfügbaren Speicherangabe auf den Bildschirm.

Normalerweise wird dieser Zeiger nicht geändert.

Es gibt aber zwei Gelegenheiten, bei denen eine Änderung dieses Zeigers sinnvoll beziehungsweise notwendig ist.

Anwendung 1:

Es kommt oft vor, daß der gesamte Speicher nicht ausschließlich für Basic benötigt wird, sondern daß ein freier Speicherbereich geschaffen wird, um zum Beispiel Maschinenprogramme, selbst definierte Zeichen oder hochaufgelöste Grafik unterzubringen, die aber nicht vom Basic-Programm überschrieben werden können.

Bei der Besprechung der Zeiger in 43 und 44 haben wir das auch schon gemacht, allerdings durch »Hochschieben« des Speicheranfangs. Mit dem Zeiger in 55 und 56 erreichen wir denselben Effekt, diesmal durch »Herunterdrücken« des Speicherendes. Gegenüber den vier Schritten beim Hochschieben ist das Herunterdrükken einfacher. Mit dem Befehl:

```
POKE 56,PEEK(56)-1:CLR
```

schieben wir das Speicherende um 256 Byte nach unten, egal für welchen Computer und welche Speichererweiterung. Mit -2 verschiebt sich das Ende um 512, mit -4 um 1 024 Byte (also 1 KByte) nach unten. Wenn Sie eine feinere Verschiebung als Vielfache von 256 benötigen, kommen Sie mit dem High-Byte in 56 allein nicht aus, sondern Sie müssen auch einen entsprechenden Wert in 55 hineinPOKEn.

Der Befehl CLR ist notwendig, denn er setzt den Zeiger der Zellen 51 und 52 (siehe dort), das heißt das untere Ende des Speicherbereichs für Zeichenketten auf dieselbe Adresse wie Zeiger 55 und 56. Dadurch wird erzwungen, daß die Zeichenkette sozusagen als Ausgangslage unterhalb des heruntergedrückten Speicherendes abgelegt wird.

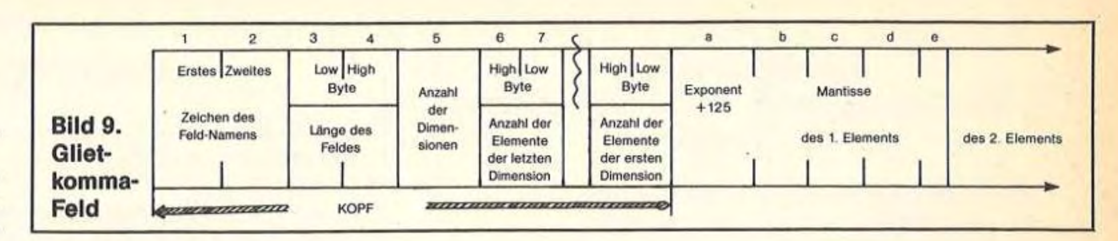


Bild 9. Gliet-komma-Feld

Anwendung 2:

Über den User-Port (Steckerleiste an der Rückseite, neben dem Datasetten-Anschluß) können VC 20 und C 64 mit anderen Geräten verbunden werden. Der Datentransfer über diese Verbindung – sie heißt RS232-Schnittstelle – muß allerdings programmiert werden. Diese RS232-Schnittstelle hat die Gerätenummer 2 (so wie der Drucker Nummer 4 und das Diskettengerät die Nummer 8 hat).

Wenn nun ein Gerät Nummer 2 mit einem OPEN-Befehl angewählt wird, wird automatisch der Zeiger in 55 und 56 und der Zeiger in 643 um 512 Byte heruntergedrückt, um je einen Eingangs- und Ausgangspufferspeicher zu erzeugen. Da der Inhalt dieser Pufferspeicher alle Variable in diesen 512 Byte überschreiben würde, wird auch der CLR-Befehl automatisch gegeben.

Es gilt daher als Vorschrift, daß bei RS232-Verbindungen zuerst der Datenkanal durch OPEN eröffnet werden muß, bevor Variable, Felder und Zeichenketten definiert werden.

Adresse 57 und 58 ($39 und $3A)

Nummer der laufenden Basic-Programmzeile

Diese Speicherzellen enthalten die Zeilennummer in Low-/High-Byte-Darstellung derjenigen Basic-Anweisung, welche gerade ausgeführt wird.

Ein kurzes Programm macht das deutlich:

```
10 PRINT "ZEILE 10",
PEEK(57)+256*PEEK(58)
20 A=3:PRINT A,PEEK(57)+
256*PEEK(58)
30 B=5:PRINT B,PEEK(57)+
256*PEEK(58)
40 PRINT A*B,PEEK(57)+
256*PEEK(58)
```

In jeder Zeile wird zuerst etwas gePRINTet, nämlich Text, Variable und ein Rechenresultat. Durch das Komma getrennt wird in der 2. Bildschirmhälfte (VC 20) beziehungsweise Bildschirmviertel (C 64) der Inhalt der Speicherzellen 57 und 58 ausgedruckt. Das Resultat zeigt

Texteinschub Nr. 10 Felder in Basic

Zur Wiederholung: Es gibt zwei Arten von Variablen, normale Variable und Felder. Jede der beiden Arten ihrerseits kann aus Gleitkomma-Zahlen, ganzen Zahlen oder Zeichenketten bestehen.

Eine normale Variable kann immer nur einen Wert haben, ein Feld enthält gleichzeitig viele Werte, alle unter demselben Variablen-Namen.

Wir können uns ein Feld mit dem Namen KARLSTRASSE als eine Liste vorstellen, in der jedes Element zwar auch den Namen Karlstraße hat, sich aber von den anderen Elementen durch eine eigene Hausnummer unterscheidet. Jede Variable in einer Hausnummer hat einen bestimmten Wert.

Während eine normale Variable einfach mit A=3 einen Wert zugewiesen bekommt, muß ein Feld erst definiert werden, nämlich wie viele Elemente es enthält. Wir machen das mit dem Befehl

```
DIM KARLSTRASSE (12)
```

Dieses Feld hat 13 Elemente (von 0 bis 12). Jedem Element kann nun ein Variablenwert zugewiesen werden durch

```
KARLSTRASSE (0)=25
KARLSTRASSE (1)=56
```

Das Feld KARLSTRASSE hat in der Klammer nur eine Zahl, man sagt, es hat nur eine Dimension.

Ein zweidimensionales Feld entspricht einem Schachbrett, mit Zahlen in der einen und Buchstaben in der anderen Dimension. Wir definieren es mit:

```
DIM AX (7,7)
```

AX ist der Name, jede Dimension hat acht Elemente, insgesamt kann das Feld 64 Werte enthalten.

Ein dreidimensionales Feld entspricht einem Quader, oder bei gleicher Elementenzahl pro Dimension (Seite) einem Würfel. Dieses wird dimensioniert mit

```
DIM BY (125,6,2)
```

Die Anzahl der Dimensionen wird nur begrenzt durch den verfügbaren Speicherplatz. Wieviel Bytes pro Feld gebraucht werden, entnehmen Sie bitte der Erklärung bei der Darstellung der Feld-Variablen (Texteinschub Nr. 11).

Ein Feld, das wie bisher gezeigt dimensioniert wird, enthält Gleitkomma-Zahlen.

Ein Feld mit ganzen Zahlen wird durch das Zeichen % nach dem Namen gekennzeichnet, also:

```
DIM CZ%(.,.,.)
```

Ein Feld mit Zeichenketten dagegen hat nach dem Namen das übliche Zeichen $, also:

```
DIM DT$(..,..,..)
```

»Wozu brauche ich Felder, wenn ich auch normale Variable verwenden kann?«, werden Sie vielleicht noch fragen.

Felder haben den großen Vorteil, daß immer dann, wenn viele Variable in einem Programm vorkommen, die alle einen gewissen Zusammenhang haben, viel Speicherplatz gespart werden kann.

Eine normale Variable braucht 7 Byte, eine Feld-Variable nur 5 oder bei ganzen Zahlen sogar nur 2 Byte. Zugegeben, vorher steht noch ein längerer Kopf, aber halt nur einmal. Und das zahlt sich bei vielen Variablen sehr rasch aus.

Und schließlich muß ich noch darauf hinweisen, daß die »Hausnummern« oder Indizes der Elemente innerhalb eines Programms durch mathematische Operationen verändert und manipuliert werden können. Aber das ist natürlich höhere Programmierkunst und geht über diese kurze Einführung hinaus.

Texteinschub Nr. 11
Darstellung der Felder-(Array)-Variablen im Speicher

Die Felder-Variablen kommen in drei Arten vor:
- als ganze Zahlen,
- als Gleitkomma-Zahlen,
- als Zeichenketten.

Sie sind in dem Texteinschub Nr. 10 »Felder in Basic« kurz beschrieben.

Wir wollen sie uns hier mit den Methoden anschauen, welche ich für den C 64 und für den VC 20 in dem Texteinschub Nr. 9 »Darstellung der normalen Variablen im Speicher« beschrieben habe.

Beim C 64 ist allerdings ein Zusatz dabei. Sie müssen, am besten gleich am Anfang, noch eingeben:

```
POKE 44,4:NEW
```

Ein eventuell auftretender SYNTAX ERROR soll uns nicht weiter stören.

Wenn Sie also das jeweilige Kochrezept ausgeführt und damit den Bildschirm- und den Variablenspeicher auf dieselbe Adresse gelegt haben, können wir anfangen.

Gleitkomma-Feld

Geben Sie direkt ein:

```
DIM AB(1,2,3)
```

Wir dimensionieren also ein Feld mit dem Namen AB, es hat drei Dimensionen, die erste Dimension hat zwei (0,1) Werte, die zweite hat drei und die dritte hat vier Werte. Sobald Sie die RETURN-Taste drücken, erscheint das Feld auf dem Bildschirm. Wir sehen folgende Zeichen:
A, B, invertiertes C, @ c @ d @ c @ b plus 120 Klammeraffen @.

Die ersten zwei Stellen sind der Name des Feldes in der Darstellung für Gleitkomma-Variable, wie in der letzten Folge beschrieben wurde. Die dritte und vierte Stelle geben im Bildschirmcode als Low- und High-Byte die Länge des Feldes an (das inverse C = 131, das @ = 0, bitte nachzählen). Die fünfte Stelle zeigt die Anzahl der Dimensionen (c = 3) an. Ab der sechsten Stelle stehen die Anzahl der Elemente der Dimension (diesmal als High- und Low-Byte) und zwar beginnend mit der letzten Dimension. In unserem Falle ist das also in Stelle 6 und 7 ein @ und d (0 - 3 =4=d), Stelle 8 und 9 sind dasselbe für die zweite Dimension und schließlich Stelle 10 und 11 für die erste Dimension (0 - 1=2=b). Danach folgen entsprechend der Anzahl der dimensionierten Elemente (2*3*4=24) 5 Byte pro Element (24*5=120), die vorerst auf 0 = @ stehen, die aber mit den Werten der Elemente aufgefüllt werden.

Dieses Auffüllen wollen wir nachvollziehen. Geben Sie bitte direkt ein:

```
AB(0,0,0)=5
```

Wir weisen damit dem allerersten Element des Feldes den Wert 5 zu.

In der oberen Darstellung des Feldes AB ändern sich dadurch Byte 12 und 13. Das neu erschienene inverse C und die Leerstelle mit den drei nachfolgenden @ ist die Gleitkomma-Darstellung (Mantisse und Exponent) der Zahl 5. Auf diese Darstellung werde ich später im Verlauf dieses Kurses bei der Besprechung der Speicherzelle 97 noch genauer eingehen.

Wenn wir jetzt (durch Überschreiben der vorigen Anweisung) zusätzlich noch eingeben:

```
AB(1,0,0)=6
```

erreichen wir eine entsprechende Änderung der Bytes 17 und 18, also des zweiten Elements des Feldes.

In Bild 9 sind die Stellen eines Gleitkomma-Feldes grafisch dargestellt.

Ganzzahliges Feld

Im Vergleich zu dem Gleitkomma-Feld dimensionieren wir als nächstes ein ganzzahliges Feld:

```
DIM AB%(1,2,3)
```

Jetzt erscheint auf dem Bildschirm gleich anschließend an das erste Feld eine neue Darstellung: invertiertes A, invertiertes B, ;, @, c@, d, @, c, @b plus 48 Klammeraffen @.

Die ersten 11 Byte haben dieselbe Bedeutung wie beim Gleitkomma-Feld, aber nur deswegen, weil wir dieselben drei Dimensionen mit identischer Elementenzahl dimensioniert haben. Bei mehr Dimensionen wäre dieser Kopf natürlich länger. Die inverse Darstellung des Feldnamens signalisiert ein ganzzahliges Feld. Die dritte Stelle zeigt das »;« - im Bildschirmcode ist das die 59. In der Tat ist das Feld nur 59 Byte lang, also wesentlich weniger als das Gleitkomma-Feld. Die 2*3*4=24 Elemente benötigen in der Ganzzahl-Darstellung nur je 2 Byte (24*2=48+11=59). Womit bewiesen ist, daß eine Ganzzahl-Darstellung mit dem Zeichen % erheblich Speicherplatz spart - allerdings nur bei Feldern!

Jetzt wollen wir noch den Inhalt des Feldes füllen, so wie vorher mit:

```
AB%(0,0,0)=5
```

... und prompt ändert sich Byte Nummer 13 in ein e (e = 5).

Eine Eingabe für das zweite Element:

```
AB%(1,0,0)=6
```

verändert das 15. Byte in ein f.

In Bild 10 ist der Inhalt eines Ganzzahl-Feldes grafisch dargestellt.

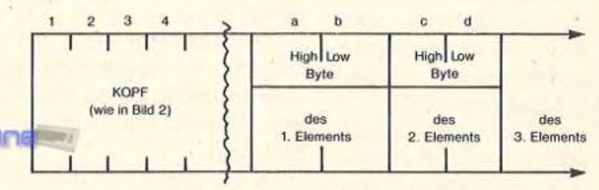


Bild 10. Ganzzahliges Feld

Felder mit Zeichenketten

Die Dimensionierung eines Feldes mit Zeichenketten sieht so aus:

```
DIM AB$(1,2,3)
```

Auf dem Bildschirm erscheint jetzt ein Feld:

Auch hier zeigen die ersten elf Stellen dieselbe Information wie bei den anderen Feldern. Zur Kennzeichnung des Zeichenketten-Feldes ist das zweite Zeichen des Feldnamens invers dargestellt. Zeichen 3 und 4 geben wieder die Länge des Feldes an. Das S hat den Bildschirmcode 83. (Vorsicht! Da wir im Groß-/Kleinbuchstaben-Modus sind, müssen wir die jeweils rechte Seite der Spalten in der Code-Tabelle nehmen). Die Länge 83 minus 11 Kopfstellen ergibt 72 Byte, geteilt durch 24 (2*3*4 = 24 Elemente) erhalten wir 3 Byte zur Darstellung eines Elements.

Das erste Byte gibt die Länge der Zeichenkette an, das zweite und dritte Byte (Low-/High-Byte) die Adresse, ab der die Zeichenkette im vierten Block gespeichert ist.

Die Methode ist also dieselbe wie bei den »normalen« Zeichenketten-Variablen. Das wollen wir uns auch noch ansehen. Geben Sie direkt ein:

```
AB$(0,0,0)="AAAAAA"
```

In der Darstellung des Feldes ändern sich dadurch die Stellen 12, 13 und 14 und wir sehen
— beim C 64:
— beim VC 20:

Im Bildschirm steht dafür:
— C 64: 6 250 159 das heißt 6 Zeichen,
ab Adresse 250+159*256=40959
— VC 20: 6 250 29 das heißt 6 Zeichen
ab Adresse 250+29*256=7674

Jetzt weisen wir dem letzten Element auch noch eine Zeichenkette zu:

```
AB$(1,2,3)="BB"
```

Die letzten drei Stellen des Feldes ändern sich ebenfalls, wobei die erste mit dem b eine Zeichenkettenlänge von 2 angibt, dementsprechend muß die Anfangsadresse um 2 niedriger sein als die vorher definierte Kette: Das Low-Byte 250 - 2 = 248, in der Codetabelle finden wir dafür das, was auch im Feld steht. Das High-Byte bleibt unverändert.

Bild 11 zeigt die grafische Darstellung des Zeichenketten-Feldes.

Als letztes zeige ich Ihnen noch die im vierten Block gespeicherten Zeichenketten. Wir drucken einfach den CHR$-Wert der in den betreffenden Speicherzellen stehenden Codezahlen aus mit:

— VC 20:

FOR I=248 TO 255:PRINT CHR$(PEEK(29*256+I));:NEXT

— C 64:

FOR I=248 TO 255:PRINTCHR$(PEEK(159*256+I));:NEXT

... und wir erhalten die beiden Zeichenketten in umgekehrter Reihenfolge, also vom Speicherende her eingespeichert. Interessant ist, daß sich vor die Felder – wenn Sie sie noch auf dem Bildschirm hatten – die neu definierte Gleitkomma-Variable I@ geschoben hat. Auch das ist eine Demonstration des Speicherverfahrens der Variablen, genauso wie ich es Ihnen in der letzten Folge erklärt habe.

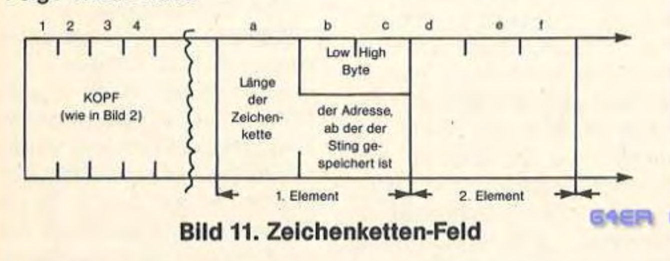


Bild 11. Zeichenketten-Feld

in der Tat die jeweilige Zeilennummer an.

Die Basic-Befehle GOTO, GOSUB-RETURN, FOR-NEXT, END, STOP, CONT und die Betätigung der STOP-Taste während eines Programmlaufes verwenden alle den Inhalt dieser Speicherzellen, um entweder zu der laufenden Zeile zurückzufinden oder um die Unterbrechung mit BREAK IN... anzuzeigen. Auch die meisten Fehlermeldungen verwenden diese Zellen.

In vielen Basic-Erweiterungen und Programmierhilfen wird ein Befehl TRACE oder STEP angeboten, welcher ein schrittweises Abarbeiten eines Programms bei gleichzeitiger Anzeige der gerade aktiven Zeilennummer erlaubt. Dieses TRACE verwendet natürlich auch den Inhalt der Zellen 57 und 58.

Schließlich sei noch erwähnt, daß im direkten Modus, also bei direkt eingetippten Aktionen des Computers ohne Programmzeilen, in der Zelle 58 immer die Zahl 255 steht. Diejenigen Basic-Befehle, welche im direkten Modus nicht erlaubt sind (INPUT, GET, DEF), prüfen in Zelle 58, ob sie im direkten Modus oder während eines Programmlaufes aufgetreten sind.

Adresse 59 und 60 ($3B und $3C)

Zeilennummer der letzten Programmunterbrechung

Immer dann, wenn ein Programmablauf durch die Befehle END oder STOP oder aber mit der STOP-Taste abgebrochen wird, wird die Nummer der gerade ausgeführten Programmzeile nach 59 und 60 gebracht und bleibt dort solange, bis eine neue Unterbrechung erfolgt.

Das läßt sich am besten mit der STOP-Taste und nachfolgendem CONT zeigen. Nehmen Sie bitte dazu das kleine Demo-Programm der Zellen 57 und 58 und ändern Sie alle PEEK-Adressen in 59 und 60 um. Fügen Sie außerdem noch eine Zeile 50 hinzu:

```
50 GOTO 10
```

Den dadurch erzeugten kontinuierlichen Lauf des Programms bremsen Sie dann mit der STOP-Taste und lassen ihn danach mit CONT weiterlaufen.

Auf der rechten Seite erscheint jetzt die Zeilennummer, bei der das Programm vorher unterbrochen worden ist.

Adresse 61 und 62 ($3D und $3E)

Zeiger auf die Adresse, ab welcher der Text der laufenden Basic-Zeile gespeichert ist.

Die Abarbeitung der einzelnen Basic-Zeilen während eines Programmlaufs wird von einem kleinen Maschinencode-Programm, welches in den Speicherzellen 115 bis 138 steht (wir kommen noch dahin), gesteuert. In den Zellen 122 und 123 enthält es die Adresse des letzten Bytes des gerade ausgeführten Basic-Befehls.

Sobald eine neue Basic-Zeile verarbeitet wird, holt das Betriebssystem diese Adresse aus 122 und 123 und speichert sie in den hier zur Diskussion stehenden Speicherzellen 61 und 62 ab, wie üblich als Low-/High-Byte.

Dasselbe geschieht bei jedem Befehl END, STOP, bei Fehlern mit dem Befehl INPUT und durch das Drücken der STOP-Taste. Der Befehl CONT hingegen schaut in 61 und 62 nach und bringt die darin befindliche Adresse zurück in die Speicherzellen 122 und 123 zur Fortsetzung des Programms. Wenn aber in Zelle 62 inzwischen eine 0 steht – und das geschieht bei einem LOAD-Befehl, durch Programm-Abbruch mit Fehlermeldung und durch Eingabe neuer Basic-Zeilen beziehungsweise deren Veränderungen mit abschließender RETURN-Taste – dann wird der CONT-Befehl nicht ausgeführt.

Zur besseren Erklärung dieser in 61 und 62 als Zeiger stehenden Adresse einer Basic-Zeile möchte ich Sie an den Texteinschub Nr. 7 erinnern, in dem ich den Basic-Programmspeicher »sichtbar« gemacht habe, um die Wirkung der Verschiebung des Zeigers in den Zellen 43 und 44 zu demonstrieren.

Wir nehmen dazu bitte noch einmal das kleine Demo-Programm für die Adressen 57 und 58 oben her und ersetzen die PEEK-Werte durch 61 und 62. Das Ausdrucken des Inhalts von 61 und 62 legen wir aber an den Anfang jeder Zeile. Das Programm sieht dann so aus:

```
10 PRINT PEEK(61)+256*PEEK
(62),"ZEILE 10"
20 PRINT PEEK(61)+256*PEEK
(62),:A=3:PRINT A
30 PRINT PEEK(61)+256*PEEK
(62),:B=5:PRINT B
40 PRINT PEEK(61)+256*PEEK
(62),A*B
```

Nach RUN erhalten wir jetzt auf der linken Seite Zahlen, die den jeweiligen Basic-Speicher angeben, ab dem diese Zeile gespeichert ist. Wenn Sie ab diesen Adressen mit der gerade erwähnten Methode aus Texteinschub Nr. 7 nachschauen, finden Sie genau die Zeilen des kleinen Demo-Programms wieder.

Zur Anwendung dieses Zeigers kann ich wenig sagen. Ihn durch POKE zu verändern, geht in Basic nicht, weil das Betriebssystem die richtigen Werte immer neu eingibt. Man kann ihn allerdings abfragen, wenn man sich für die Speicheradressen der Basic-Zeilen interessiert. Die einzige Anwendung dafür kenne ich von S. Leemon, welche bei den Adressen 65 und 66 eingesetzt wird.

Adresse 63 und 64 ($3F und $40)

Zeilennummer eines gerade laufenden DATA-Befehls

Diese Speicherzellen enthalten die Nummer der Basic-Zeile, in der gerade ein DATA-Befehl mit READ gelesen wird. Sobald in einer DATA-Zeile ein Fehler gefunden wird, kommt diese Zeilennummer aus 63 und 64 in die Speicherzellen 57 und 58, um in der Fehlermeldung die fehlerhafte DATA-Zeile und nicht die laufende READ-Zeile anzuzeigen. Auf diese Weise werden Syntax-Fehler in einer DATA-Zeile angezeigt. Um andere Fehler, wie zum Beispiel ein fehlendes Komma zwischen zwei DATA-Angaben anzuzeigen, können die Speicherzellen 63 und 64 eingesetzt werden.

In dem folgenden Programm wird in Zeile 20 geprüft, ob die DATA-Angaben größer als 255 sind. Da bei einem fehlenden Komma die beiden Zahlen als eine Zahl gelesen werden, wird dieser Fall erkannt und mit einem F versehen die Nummer der DATA-Zeile ausgedruckt, in der das Komma fehlt.

```
10 FOR X=1 TO 10:READ
A:PRINT A
20 IF A>255 THEN PRINT
"F" PEEK(63) + 256*PEEK
(64)
30 NEXT X
40 DATA 10,20,30
50 DATA 40,50,60
60 DATA 70,80,90,100
```

Sie können jetzt in den DATA-Zeilen Kommafehler einbauen, die vom Programm angezeigt werden. Ein anderer häufiger Fehler, nämlich ein Komma am Ende einer DATA-Zeile, kann damit leider nicht erkannt werden. Aber vielleicht fällt Ihnen eine Prüfformel dazu ein.

Adresse 65 und 66 ($41 und $42)

Zeiger auf die Adresse, ab der die laufende DATA-Angabe gespeichert ist

Diese Speicherzellen enthalten in der Low-/High-Byte-Darstellung die Adresse im Basic-Programmspeicher, ab welcher der READ-Befehl nach der nächsten DATA-Zeile sucht.

Zu Beginn eines Programms steht in 65 und 66 als Adresse der Beginn des Basic-Speichers, also derselbe Wert wie in den Speicherzellen 43 und 44. Der Befehl RESTORE setzt den Zeiger immer auf diesen Anfangswert zurück. Ein Demo-Programm zeigt uns das an (die Kommata sind wichtig für das Format der Darstellung auf dem Bildschirm!):

```
10 PRINT, PEEK(65)+256*
PEEK(66)
20 FOR X=1 TO 10:READ A
30 PRINT A,PEEK(65)+
256*PEEK(66)
40 NEXT X
50 DATA 10,20,30,40,50,60,
70,80,90,100
60 RESTORE
70 PRINT,PEEK(65)+256*PEEK
(66)
```

Durch Verändern dieses Zeigers in 65 und 66 kann die Reihenfolge, mit der DATA-Angaben gelesen werden, verändert werden, allerdings nur zeilenweise.

Wir brauchen dazu die oben beschriebenen Speicherzellen 61 und 62, deren jeweiligen Inhalt wir ja mit PEEK abfragen können. Wenn wir das vor jeder DATA-Zeile machen und diesen Wert einer Variablen zuweisen, haben wir die Adresse gespeichert, hinter welcher die DATA-Zeile kommt. Durch POKEn dieser Adressen in die Speicherzellen 65 und 66 vor einem READ-Befehl, wird diesem READ die nächste DATA-Zeile vorgegeben und wir können so die Reihenfolge der DATA-Zeilen ändern.

```
10 A1=PEEK(61)+PEEK(62)*256
20 DATA DAS IST DIE 1. ZEILE
30 A2=PEEK(61)+PEEK(62)*256
40 DATA DAS IST DIE 2.ZEILE
50 A3=PEEK(61)+PEEK(62)*256
60 DATA DAS IST DIE 3.ZEILE
70 POKE 65,A3 AND 255:
POKE 66,A3/256:
READ A$:PRINT A$
80 POKE 65,A1 AND 255:
POKE 66,A1/256:
READ A$:PRINT A$
90 POKE 65,A2 AND 255:
POKE 66,A2/256:
READ A$:PRINT A$
```

Mit den Zeilen 70 bis 90 werden für jede DATA-Zeile eigene READ-Anweisungen gegeben. Welche DATA-Zeile gelesen werden soll, wird durch die Variablen Ax und Bx (x=1,2,3) bestimmt, mit denen der Zeiger in 65 und 66 »verbogen« wird. Auf ein Detail will ich hier hinweisen:

Die Adresse 61 und 62 darf nicht mit zwei Befehlen, sondern muß mit einem Befehl ausgelesen werden, da bei einem möglichen Page-Wechsel zwischen den zwei Befehlen der Zeiger nicht verbogen, sondern abgeknickt wird.

Was passiert in der ersten Zeile des Demo-Programms?

```
10 A1=PEEK(61):B1=PEEK(62)
```

Mit »A1=PEEK(61)« wird der Variablen A1 der Wert des Low-Bytes des Zeigers 61 und 62 zugewiesen. Dieser zeigt am Anfang einer Zeile auf das Null-Byte vor der Linkadresse (hier 2048), so daß A1 den Wert (2048 AND 255)=0 erhält. Mit »B1=PEEK(62)« wird der Variablen B1 der Wert des High-Bytes des Zeigers 61 und 62 zugewiesen. Dieser zeigt aber inzwischen auf das Trennzeichen (»:«) zwischen den beiden Befehlen (hier 2061), so daß B1 den Wert (INT(2061/256))=8 erhält. Als Zeiger auf das aktuelle DATA-Element erhalten wir die erwartete Adresse (A1+B1*256)=2048.

Was aber, wenn Zeilenanfang und Trennzeichen nicht in derselben Page liegen? Dazu setzen Sie bitte den Basic-Anfang um eine Stelle zurück:

```
POKE43,0:POKE 2047,0:NEW
```

Die Zeiger auf den Zeilenanfang und das Trennzeichen werden dadurch ja ebenfalls verändert, so daß A1 jetzt den Wert (2047 AND 255)=255 und B1 den Wert (INT(2060/256))=8 erhält. Als Zeiger auf das aktuelle DATA-Element erhalten wir nun die völlig unbrauchbare Adresse (A1+B1*256)=2303.

Adresse 67 und 68 ($43 und $44)

Zeiger auf die Adresse, aus welcher die Befehle INPUT, GET und READ die Zeichen/Zahlen holen

INPUT und GET verlangen Angaben, die per Tastatur eingegeben werden. Tastatur-Eingaben im direkten Modus, also wenn kein Programm läuft, werden im Eingabe-Pufferspeicher des Editors (der Teil des Betriebssystems, welcher für die Zeilendarstellung auf dem Bildschirm verantwortlich ist) ab Speicherzelle 512 bis 600 zwischengespeichert.

Der Zeiger in 67 und 68 zeigt auf die jeweilige Adresse in diesem Eingabe-Pufferspeicher. Bei READ ist 67 und 68 identisch mit 65 und 66. Der Inhalt dieser Speicherzellen kann mit PEEK ausgelesen werden.

Adresse 69 und 70 ($45 und $46)

Name der gerade aufgerufenen Basic-Variablen

Wenn beim Ablauf eines Programms eine Variable auftaucht, muß ihr derzeitiger Wert im Variablen-Speicher gesucht werden. Während dieses Suchvorgangs wird der Name der Variablen in 69 und 70 zwischengespeichert. Die Form der Zwischenspeicherung ist dieselbe 2-Byte-Darstellung wie im Variablenspeicher, beschrieben bei der Behandlung der Speicherzellen 45 und 46.

Adresse 71 und 72 ($47 und $48)

Zeiger auf die Adresse des Wertes der gerade aufgerufenen Basic-Variablen

Ähnlich wie bei 69 und 70 wird hier während des Anrufes einer Variablen durch ein Programm ein Wert zwischengespeichert, diesmal aber nicht der Name der Variablen, sondern der 2-Byte-Wert, welcher direkt hinter dem Variablennamen steht. Nähere Einzelheiten sind im Text der Speicherzellen 45 und 46 beschrieben.

Davon ausgenommen sind selbstdefinierte Funktionen. Wie im Texteinschub Nr. 12 »Darstellung der Variablen einer selbstdefinierten Funktion« gezeigt ist, erscheinen diese ebenfalls im Variablenspeicher in einer Darstellung, welche den normalen Variablen sehr ähnlich ist.

Damit nun eine normale oder Feld-Variable denselben Namen haben kann wie eine Funktion, wird die oben genannte Zwischenspeicherung in 69 und 70 bei Funktionen unterdrückt.

Adresse 73 und 74 ($49 und $4A)

Zwischenspeicher für Variable einer FOR-NEXT-Schleife und für diverse Basic-Befehle

Die Adresse einer Schleifenvariablen wird zuerst hier gespeichert, bevor sie auf den Stapelspeicher ab Speicherzelle 256 ($100) gebracht wird. Die Funktion und Arbeitsweise des Stapelspeichers werden wir bei diesen Adressen behandeln. Etliche Basic-Befehle, wie LIST, WAIT, GET, INPUT, OPEN, CLOSE und andere, verwenden die Speicherzellen 73 und 74 für Zwischenspeicherungen. Diese Adressen sind für den Basic-Programmierer daher nicht verwendbar.

Adresse 75 und 76 ($4B und $4C)

Zwischenspeicher für Zeiger bei READ und mathematischen Operationen

Während der Auswertung eines mathematischen Ausdrucks durch die Routine FRMEVL des Basic-Übersetzers, wird der Platz des betroffenen mathematischen Operators in einer Tabelle, hier in 75 und 76, zwischengespeichert. Dieser Platz wird dabei als Abstand zum Beginn der Tabelle dargestellt. Außerdem verwendet der READ-Befehl diese Adressen als Zwischenspeicher für einen Programmzeiger. Die Speicherzellen 75 und 76 sind in Basic nicht verwendbar.

Adresse 77 ($4D)

Hilfsspeicher für Vergleichs-Operationen

Die bei 75 und 76 schon erwähnte Auswertungs-Routine FRMEVL erzeugt in der Speicherzelle 77 einen Wert, der angibt, ob es sich bei einer Vergleichsoperation um den Fall »kleiner als« (<), »gleich wie« (=) oder »größer als« (>) handelt. Diese Speicherzelle ist nur im Maschinencode erreichbar.

Adresse 78 und 79 ($4E und $4F)

Zeiger auf Adresse, ab welcher der Wert der Variablen einer selbstdefinierten Funktion gespeichert ist

Basic erlaubt es bekanntlich, mit dem Befehl DEF selbst erfundene Funktionen zu definieren, welche die Form FN gefolgt von einem Variablennamen haben, zum Beispiel

```
DEF FNAA(X).
```

Im Texteinschub Nr. 12 »Darstellung von Variablen selbstdefinierter Funktionen« wird gezeigt beziehungsweise sichtbar gemacht, wie derartige Funktionen und ihre Variablen gespeichert werden. Während der Definition einer Funktion steht in 78 und 79 die Adresse, ab welcher die Funktion und der Wert ihrer Variablen gespeichert ist. Der Inhalt dieser Adressen ist identisch mit den Zeichen hinter dem Namen der Funktion (1. Gruppe im nebenstehenden Beispiel).

Nach der Ausführung der Funktion steht in 78 und 79 allerdings die Adresse, ab welcher der Zahlenwert der Funktion selbst gespeichert ist. Er ist identisch mit den Zeichen der 2. Gruppe.

Diesen Zusammenhang können Sie überprüfen, indem Sie im Programm des Texteinschubes folgende Zeilen hinzufügen:

```
25 PRINT PEEK(78)+256*
PEEK(79)
35 PRINT PEEK(78)+256*
PEEK(79)
```

Nach RUN erhalten Sie zwei Adressen, die Sie mit direkter Eingabe abfragen:

```
FOR I=0 TO 4:PRINT PEEK
(1.Adresse+I);:NEXT I:
FOR J=0 TO 4:PRINT PEEK
(2.Adresse+J);:NEXT J
```

Sie werden sehen, daß der Inhalt der beiden Adressen genau den Werten der Zeichen 3 bis 7 der beiden Gruppen entspricht, allerdings im Bildschirmcode.

Adresse 80 bis 82 ($50 bis $52)

Zeiger auf einen provisorischen Speicherplatz einer Zeichenkette, die gerade bearbeitet wird

Die Teilprogramme (von Programmierern »Routinen« genannt) des Basic-Übersetzers im ROM des Computers, welche Zeichenketten (Strings) behandeln, verwenden die ersten beiden Bytes dieser drei Speicherzellen, nämlich 80 und 81, um in Low-/High-Byte-Darstellung diejenige Speicheradresse anzugeben, ab der die Zeichenkette im Programmspeicher zu finden ist.

Das dritte Byte (82) enthält die Länge der Zeichenkette. Wegen der provisorischen Natur dieses Zeigers ist er für Basic-Programme nicht geeignet.

Adresse 83 ($53)

Flagge für die Garbage Collection

In dieser Speicherzelle steht während der sogenannten Garbage Collection (Müllabfuhr) eine Zahl, die angibt, ob die Variable der zur Überprüfung anstehenden Zeichenkette eine Länge von 3 oder 7 Byte hat.

Der Vorgang der Garbage Collection ist von B. Schneider, 64'er-Ausgabe 1/85, ausführlich beschrieben worden. Angaben über die Bedeutung der Variablen einer Zeichenkette finden Sie in den Texteinschüben Nr. 9 und Nr. 11.

Adresse 84 bis 86 ($54 bis $56)

Sprungbefehl auf die Adressen der Basic-Funktionen

Jede Basic-Funktion, wie zum Beispiel SGN, INT, ABS, USR und so weiter, wird durch ein spezielles Teilprogramm (Routine) des Basic-Übersetzers ausgeführt. Die Anfangsadresse jeder dieser Routinen sind in einer Tabelle im ROM fest eingespeichert. Im VC 20 steht diese Tabelle von 49234 bis 49279 ($C052 bis $C07F), im C 64 von 41042 bis 41087 ($A052 bis $A07F).

In der Speicherzelle 84 steht der Sprungbefehl JMP in Maschinencode, dargestellt

---

Texteinschub Nr. 12
Darstellung der Variablen einer selbstdefinierten Funktion

Ich habe Ihnen gezeigt, wie im Programmspeicher abgelegte normale Variablen und Felder-Variablen sichtbar gemacht werden können. Damit konnten wir den Aufbau und die Darstellung der einzelnen Variablenarten studieren.

Heute will ich einen weiteren Variablentyp vorstellen, nämlich den der selbstdefinierten Funktionen.

Sie erinnern sich vielleicht, mit dem Basic-Befehl »DEF FN (Name)(Variable)« können wir komplizierte Funktionen selbst erfinden, definieren und später als »FN (Name)(Variable)« weiter verarbeiten. Diesen Typ wollen wir uns anschauen, wie er im Speicher steht.

Im Prinzip verwenden wir dieselben Methoden zur Sichtbarmachung wie die letzten Male.

Aber ein Unterschied kommt noch dazu. Der Befehl DEF kann leider nicht direkt eingegeben werden, sondern muß immer als Teil einer Programmzeile mit einer Zeilennummer versehen sein.

Deshalb schreiben wir zuerst ein kleines Programm zur Definition der Funktion plus Variable, bevor wir den Variablenspeicher mit dem Bildschirmspeicher zusammenlegen:

```
10 DEF FNAA(X)=3*SIN(X)+COS(X)
20 X=5
30 PRINT FNAA(X)
```

Die Funktion hat also den Namen »AA«. Bevor wir weitermachen, überprüfen Sie bitte mit RUN, ob alles stimmt. Nun wird der Speicher verschoben.

Für den C 64 gilt:

1. POKE 46,4:POKE 48,4
2. Bildschirm löschen mit CLR-Taste
3. Cursor auf die Mitte fahren
4. LIST (es erscheint das Programm)
5. auf den 2. Zeichensatz umschalten (mit C= und SHIFT-Taste)
6. RUN

Für den VC 20 (ohne Erweiterung) gilt:

Nur den Bildschirm auf 4096 zu verschieben, wie das letzte Mal, geht diesmal nicht, da wir ja für DEF ein kleines Programm schreiben müssen.

Also legen wir Bild- und Variablenspeicher ab Adresse 5120 (5120/256=20).

1. POKE 46,20:CLR
2. POKE 648,20
3. STOP/RESTORE-Tasten, bis Cursor wieder da ist
4. Bildschirm löschen mit CLR-Taste
5. die ersten vier bis sechs Zeilen mit SPACE-Taste überfahren
6. Cursor ein paar Zeilen nach unten
7. LIST (es erscheint das Programm)
8. mit Commodore- und SHIFT-Taste auf 2. Zeichensatz umschalten
9. RUN

Wir sehen jetzt oben zwei Gruppen mit je sieben Zeichen, wie üblich.

Die erste Gruppe stellt die Funktion FNAA(x) dar. Sie ist gekennzeichnet durch das invertierte erste Zeichen des Namens, während das zweite Zeichen normal erscheint.

Das dritte und vierte Zeichen gibt in Low-/High-Byte-Darstellung (im Bildschirmcode) die Adresse an, ab der die Funktion FNAA(x) im Programmspeicher abgelegt ist. Mit `PEEK (3. Zeichen)+256*PEEK (4. Zeichen)` kann das abgefragt werden.

Das fünfte und sechste Zeichen nennt die Adresse, an welcher der Zahlenwert der Funktions-Variablen X anfängt. Das siebente Zeichen schließlich ist das erste Zeichen der Funktion selbst (in unserem Beispiel die 3).

Die zweite Gruppe beschreibt die Variable X der Funktion. Die normale Darstellung der beiden ersten Zeichen, die den Namen darstellen, gibt uns an, daß es sich um eine Gleitkomma-Variable handelt, deren Wert als Mantisse und Exponent dargestellt ist.

Der Aufbau einer Funktion ist in Bild 12 zusammengefaßt:

1	2	3	4	5	6	7
Erstes	Zweites	Low-	High-	Low-	High-	
Zeichen des Funktionsnamens		Byte der Adresse, ab der die Funktion abgespeichert ist		Byte der Adresse, ab dem der jeweilige Wert der Funktionsvariablen X abgespeichert ist		1. Zeichen der Funktion
ASCII-Wert + 128	ASCII-Wert					

Bild 12. Selbstdefinierte Funktion

durch die Zahl 75 ($4C). In den beiden anderen Zellen 85 und 86 steht dann in Low-/High-Byte-Darstellung die jeweilige Adresse in der Tabelle, welche der vom Programm gerade gebrauchten Basic-Funktion entspricht. Dieser gesamte Befehl JMP plus Adresse entspricht in Basic der GOSUB-Zeilennummer.

Ein Beispiel soll das verdeutlichen. Geben Sie direkt ein:

```
PRINT PEEK(84);PEEK(85);
PEEK(86)
```

Wir erhalten

beim C 64: 76 13 184

beim VC 20: 76 13 216

Die erste Zahl ist genauso wie oben beschrieben. Die beiden anderen Zahlen ergeben zusammen die Adresse 47117 ($B80D) beziehungsweise 55309 ($D80D). Wenn Sie ein Buch mit ROM-Listing haben, werden Sie unter dieser Adresse die Routine für die Funktion »PEEK« finden. Das ist natürlich nicht erstaunlich, haben wir doch gerade vorher als letzten Befehl genau diese Funktion eingegeben.

Leider ist das auch die einzige Funktion, die ich Ihnen vorführen kann, denn zum Vorführen muß ich eben immer PEEKen, so daß beim besten Willen immer nur die oben angegebenen Zahlen erscheinen können.

Adresse 87 bis 96 ($57 bis $60)

Arbeitsspeicher für diverse Arithmetik-Routinen des Basic-Übersetzers

Diese zehn Speicherplätze werden von verschiedenen Teilprogrammen (Routinen), besonders bei arithmetischen Operationen, als Zwischenspeicher verschiedener Werte, Flaggen und Zeiger benutzt.

Adresse 97 bis 102 ($61 bis $66)

Gleitkomma-Akkumulator Nr.1

»Akkumulator« heißt seit der Zeit der mechanischen Rechenmaschinen eine Speicherzelle, welche bei Rechenoperationen dadurch im Mittelpunkt steht, daß laufend Daten in sie hineingeschrieben beziehungsweise aus ihr herausgelesen werden.

Normalerweise trägt diesen Namen das zentrale Rechenregister des Mikroprozessors. Leser des Assembler-Kurses kennen diesen Akkumulator inzwischen zur Genüge.

Die Speicherzellen 97 bis 102 werden deswegen ebenfalls Akkumulator genannt, weil sie bei der Verarbeitung von Gleitkommazahlen eine ähnliche zentrale Rolle spielen.

Zelle 97 enthält den Exponenten. Die Zellen 98 bis 101 enthalten die Mantisse.

Zelle 102 enthält das Vorzeichen der Gleitkommazahl. Eine 0 bedeutet ein positives, die Zahl 255 ein negatives Vorzeichen.

Mit dem Gleitkomma-Akkumulator Nr. 1 sind zwei weitere Speicherzellen eng verbunden, nämlich 104 ($68) und 112 ($70).

Ganz zum Schluß ist noch erwähnenswert, daß nach der Umwandlung einer Gleitkommazahl in eine ganze Zahl diese als Low-/High-Byte in den beiden Speicherzellen 98 und 99 steht, was für Maschinenprogramme vielleicht recht nützlich sein kann.

Adresse 103 ($67)

Zwischenspeicher beziehungsweise Zählregister

Diese Adresse wird von zwei Routinen verwendet. Der Basic-Übersetzer benutzt sie als Vorzeichenspeicher bei der Umwandlung von Zahlen aus dem ASCII-Format in Gleitkommazahlen. Das Betriebssystem verwendet diese Adresse als Zähler der Abarbeitungsschritte bei der Berechnung eines Polynoms der Form

y=a0+a1*x+a2*x↑2+a3*x↑3+.....

Adresse 104 ($68)

Überlauf-Speicher des Gleitkomma-Akkumulators Nr. 1

Wenn eine Zahl so groß wird, daß sie mit den zur Verfügung stehenden Stellen nicht mehr dargestellt werden kann, sprechen wir von einem »Überlauf«.

Bei Gleitkommazahlen liegt diese Überlaufgrenze bei $1,70141183 * 10^{38}$.

Während einer mathematischen Berechnung kann es intern im Computer vorkommen, daß ein Überlauf eintritt, der aber am Ende der Operation wieder verschwinden würde. Der Akkumulator Nr. 1 benutzt in einem derartigen Fall die Speicherzelle 104, um die verfügbare Stellenzahl um 8 Bit zu vergrößern. Für endgültige Resultate steht diese Erweiterung natürlich nicht zur Verfügung.

Dieser Vorgang tritt besonders häufig bei der Umwandlung von ganzen Zahlen oder Zeichenketten in Gleitkommazahlen auf.

Adresse 105 bis 110 ($69 bis $6E)

Gleitkomma-Akkumulator Nr. 2

Spätestens jetzt verstehen Sie, warum der Akkumulator der Speicherzellen 97 bis 102 die Nr. 1 hat. Es gibt hier noch einen zweiten Gleitkomma-Akkumulator, der ein identischer Zwilling ist. Zwei Akkumulatoren sind immer dann notwendig, wenn mathematische Operationen ablaufen, welche mehr als einen Operanden verarbeiten, wie zum Beispiel Multiplikation, Division und so weiter.

Aufgrund der Identität der beiden Akkumulatoren kann ich mir eine weitere Beschreibung ersparen.

Adresse 111 ($6F)

Flagge für Vorzeichenvergleich der Gleitkomma-Akkumulatoren Nr. 1 und Nr. 2

Wenn die Zahl in beiden Akkumulatoren gleiche Vorzeichen hat, steht in Speicherzelle 111 eine 0, bei verschiedenen Vorzeichen eine 255.

Adresse 112 ($70)

Rundungsspeicher des Gleitkomma-Akkumulators Nr. 1

Es kann vorkommen, daß die Mantisse einer Gleitkommazahl mehr Stellen hat, als mit den vier Mantissen-Bytes des Akkumulators Nr. 1 (Zelle 97 bis 102) dargestellt werden können. In diesem Fall werden die hintersten, das heißt die unwichtigsten Stellen hinter dem Komma in der Zelle 112 abgelegt. Von dort werden sie geholt, um die Genauigkeit von mathematischen Operationen zu erhöhen und auch um Endresultate abrunden zu können.

Adresse 113 und 114 ($71 und $72)

Zwischenspeicher für verschiedene Routinen

Diese Speicherzellen werden von sehr vielen Routinen des Übersetzers und des Betriebssystems, wie zum Beispiel Zeichenkettenverarbeitung, interne Uhr (TI$), Bestimmung der Größe von Feldern (Arrays) und etlichen anderen verwendet.

Adresse 115 bis 138 ($73 bis $8A)

Teilprogramm »Nächstes Zeichen eines Basic-Textes holen« (CHRGET-Routine)

Die Problematik der Übersetzung von Basic-Befehlen und Anweisungen besteht darin, daß die Übersetzungsschritte durch entsprechende Programmteile des Basic-Übersetzers im Computer fest vorprogrammiert sein müssen, was bedeutet, daß diese Programme natürlich im – nicht veränderbaren – ROM stehen.

Auf der anderen Seite verlangt aber der Übersetzungsvorgang, daß gewisse Teile dieser Programme sich laufend verändern. Als Beispiel soll der Zeiger herhalten, der angibt, in welcher Speicherzelle das nächste zu bearbeitende Zeichen steht. Dieser Zeiger und die zusammengehörigen Programmschritte dürfen natürlich nicht im ROM stehen, denn da sind sie ja nicht änderbar.

Dieser Konflikt wird dadurch gelöst, daß dieses »variable« Teilprogramm des Übersetzers zwar im ROM steht (im C 64 ab 58274 oder $E3A2, im VC 20 ab 58247 oder $E387), von wo es aber direkt nach dem Einschalten des Computers in das RAM, und zwar in die Speicherzellen 115 bis 138, umgeladen wird.

Dieses Teilprogramm, welches die Zeichen zur Übersetzung herbeiholt und deswegen »Character-Get« oder kurz CHARGET-Routine genannt wird, ist wegen seiner Veränderbarkeit natürlich ein beliebtes Objekt aller möglichen Manipulationen. Es ist deshalb im Assembler-Kurs, Teil 5, im 64'er, Ausgabe 1/85, im Detail beschrieben worden, allerdings mit Schwerpunkt auf Assembler-Maschinensprache.

Für Basic-Programmierer möchte ich hier deshalb eine kurze Beschreibung der CHARGET-Routine einfügen.

Die Routine beginnt mit einem Sprung auf den oben schon erwähnten Zeiger in Adresse 122 und 123, welcher seinerseits auf die Adresse zeigt, in welcher das nächste zu übersetzende Zeichen steht. Das Zeichen wird entsprechend dem Hinweis des Zeigers geholt, in den Akkumulator des Mikroprozessors geladen und dort verschiedenen Prüfungen unterzogen. Ist das Zeichen ein Gänsefuß, erkennt das Programm, wie es das nächste Zeichen interpretieren und behandeln muß. Ein Doppelpunkt leitet einen neuen Befehl ein, eine Leerstelle wird unterdrückt und so weiter.

Mit dem Befehl

```
PRINT PEEK(122)+256*
PEEK(123)
```

können wir innerhalb eines Programms ausdrucken, wohin der Zeiger nach dem letzten Basic-Zeichen deutet. Eine Überprüfung mit den Methoden, die ich bei der Besprechung der Speicherzellen 43 bis 56 genannt habe, zeigt Ihnen den Zusammenhang.

Normalerweise wird der Zeiger in 122 und 123 nach jedem Zeichen um 1 erhöht, da ja die Zeichen einer Basic-Zeile hintereinander im Speicher stehen. Ein GOTO- oder GOSUB-Befehl kann diese Folge natürlich unterbrechen, ebenso wie eine willkürliche Änderung durch einen Eingriff von außen.

Ein derartiger Eingriff, auch »wedge« (Keil) genannt, öffnet natürlich Tür und Tor für Programmiertricks, insbesondere für Einbau von neuen, selbsterfundenen Befehlen. Man kann entweder den allerersten Sprungbefehl auf den Zeiger so umlenken, daß er auf ein eigenes Maschinenprogramm springt, oder man kann den Zeiger selbst »verbiegen«, so daß er auf eine andere Adresse und damit auf ein anderes Zeichen zeigt. Es gibt dafür viele Möglichkeiten, die aber alle nur in Maschinencode funktionieren. Theoretisch können wir natür-

Texteinschub Nr. 13
Wie zufällig sind Zufallszahlen?

Der Befehl RND(X) ergibt eine Zufallszahl zwischen 0 und 1 – so steht es im Commodore-Handbuch.

Eine Zufallszahl ist definitionsgemäß rein dem Zufall überlassen. Ihr Wert kann nicht vorhergesehen werden. Wie kann aber ein Computer, in dem alle Vorgehensweisen und Arbeitsschritte fest vorprogrammiert sind, eine zufällige Zahl erzeugen? Die Commodore-Computer machen das so:

Der Befehl RND nimmt eine bestimmte Ausgangszahl (auf die ich noch näher eingehen werde), auf englisch »seed« = Samen genannt, multipliziert sie mit 11879546.4 und zählt $3.92767778 * 10^8$ dazu. Die 5 Byte der resultierenden Gleitkommazahl werden miteinander vertauscht und in einen positiven Bruch umgewandelt. Diese Manipulation ergibt die »Zufallszahl«, die als neuer »Samen« in den Speicherzellen 139 bis 143 gespeichert wird.

Es ist sicher einzusehen, daß die Zufälligkeit nicht sehr hoch sein kann, es sei denn, die oben genannte und noch nicht erklärte Ausgangszahl ist zufällig.

Die erste Ausgangszahl hängt vom »Argument« des RND(X)-Befehls ab, das heißt vom Wert X, der in der Klammer dahinter steht. Es gibt drei Möglichkeiten für das Argument:
- eine positive Zahl (egal, welcher Wert)
- eine negative Zahl
- die Zahl 0

Eine positive Zahl

zum Beispiel RND (1) oder RND(56) nimmt als Samen die Zahl 0.811635157, die beim Einschalten des Computers als 5-Byte-Gleitkommazahl in die Speicherzellen 139 bis 143 geschrieben worden ist. In den fünf Zellen stehen die Zahlen 128, 79, 199, 82, 88.

Daraus folgt aber, daß nach dem Einschalten des Computers mit RND(1) immer dieselbe Sequenz von Zufallszahlen erzeugt wird. Schalten Sie bitte den Computer aus und ein und geben Sie ein:

```
10 PRINT RND(1):GOTO 10
```

Notieren Sie die ersten paar Zahlen und wiederholen Sie mit Aus-/Einschalten die Prozedur. Sie werden immer dieselben Zahlen sehen.

Zum Austesten von Programmen mit bekannten Zahlensequenzen ist diese Methode sicher wichtig, aber echte Zufallszahlen sind das nicht!

Eine negative Zahl

zum Beispiel RND(-1) oder RND(-95) bringt als erstes das Argument selbst (in meinem Beispiel -1 oder -95) als Gleitkommazahl in die Speicherzellen 139 bis 143, von wo sie als Samen den schon beschriebenen Manipulationen unterworfen wird. Nur – mit einem bestimmten negativen Argument erhalten Sie immer dieselbe Zufallszahl. Probieren Sie es aus:

PRINT RND(-2) ergibt immer dieselbe Zahl.

Es mag Fälle geben, wo die Vorgabe des allerersten Samens wünschenswert ist. Ich will aber von zufälligen Zahlen sprechen. Wir können diese Methode des negativen Arguments dadurch verbessern, daß wir als Argument selbst eine Zufallszahl nehmen.

Als derartige Zahl bietet sich der Wert der inneren Uhr TI an, die beim Einschalten des Computers losläuft und 60mal in der Sekunde weitergestellt wird. Da kein Mensch wissen kann, welchen Wert die Uhr TI gerade hat, kommt der Befehl RND(-TI) dem absoluten Zufall schon sehr nahe.

Das Argument (0)

verwendet eine andere Methode. Als Samen nimmt er eine sich ständig ändernde Zahl, die beim VC 20 aus vier Registerinhalten des VIC-Interface-Bausteins genommen werden. Beim C 64 wird es ähnlich gemacht, nur ist der VIC-Baustein ein anderer Typ.

Mit derselben Methode nach dem Einschalten wie im ersten Fall oben können Sie das leicht überprüfen.

Ich habe eingangs zitiert, daß RND(X) eine Zahl zwischen 0 und 1 erzeugt; das gilt aber nur für ein positives Argument. Wenn Sie hingegen eine Zufallszahl innerhalb eines ganz bestimmten Bereiches brauchen, müssen Sie anders vorgehen.

Kochrezept Nr. 1

Mit folgender Formel ist der Zahlenbereich beliebig vorgebbar:

```
X=(RND(1)*A)+B
```

Die Zahl A gibt einen Bereich von 0 bis A vor.

Die Zahl B legt den untersten Wert des Bereiches fest.

Beispiele:

`10 PRINT (RND(1)*6)+1:GOTO 10` erzeugt Zahlen von 1 bis 6

`10 PRINT (RND(1)*52)+1:GOTO 10` erzeugt Zahlen von 1 bis 52

`10PRINT (RND(1)*6)+10:GOTO 10` erzeugt Zahlen von 10 bis 16

Mit dem Vorschalten der Funktion INT vor den Befehl RND werden die Zufallszahlen auf ganze Zahlen beschränkt.

```
10 PRINT INT (RND1)*6)+10:GOTO 10
```

Noch einmal: Zufallszahlen innerhalb bestimmter Zahlenbereiche sind gekoppelt mit einem positiven Argument. Wir haben aber gesehen, daß gerade so keine echten Zufallszahlen erzeugt werden. Deshalb brauchen wir noch ein zweites Kochrezept.

Kochrezept Nr. 2

Wenn Sie in einem Programm nach dem Einschalten des Computers immer neue Zufallszahlen brauchen, ist es empfehlenswert, für die allererste Zufallszahl RND(-TI) oder RND(0) zu verwenden, dann aber mit RND(1) fortzufahren.

Dasselbe gilt, wenn ein Programm wegen INPUT oder WAIT eine Pause hat. Nach der Pause sollte zuerst ein neuer Ausgangswert genommen werden.

Als letztes will ich noch beschreiben, wie man Zufallszahlen innerhalb von Maschinenprogrammen erzeugen kann.

Im Betriebssystem steht natürlich eine Routine für den Befehl RND. Im C 64 beginnt sie ab 57495 ($E097), im VC 20 ab 57492 ($E094).

Der Ausgangswert (Samen) wird dabei aus dem Gleitkomma-Akkumulator Nr. 1 geholt, dessen Vorzeichen oder Wert 0 das weitere Vorgehen der Routine bestimmt.

Sie müssen also den Samen in den Akkumulator Nr. 1 laden und dann mit JSR auf die RND-Routine springen. Als Resultat können Sie einen oder mehrere Werte der Zellen 140 bis 143 verwenden und nach Belieben weiterverarbeiten.

lich den Inhalt des Zeigers in 122 und 123 durch POKE verändern. Aber was dann? Jeder nachfolgende Basic-Befehl löst natürlich wieder die normale Übersetzungsroutine aus und unser schöner POKE ist für die Katz.

Wie ein Wedge in Maschinensprache gemacht wird, hat Christoph Sauer im VC 20-Kurs – 64'er, Ausgabe 9/84 – beschrieben. Allerdings ist das Beispiel für Anfänger nicht verständlich, was mich zu der Überzeugung bringt, daß die CHARGET-Routine und ihre Anwendung einen eigenen Aufsatz wert wäre.

Adresse 139 bis 143 ($8B bis $8F)

Wert der RND-Funktion als Gleitkommazahl

Mit dem Befehl RND(X) kann bekanntlich eine Zufallszahl erzeugt werden. Was das bedeutet und wie »zufällig« diese Zahlen sind, können Sie dem Texteinschub Nr. 13 »Wie zufällig sind Zufallszahlen?« entnehmen.

Beim Einschalten des Computers werden die Zahlen 128, 79, 199, 82 und 88 in diese Speicherzellen geschrieben. Mit der folgenden Zeile können Sie das gleich nach dem Einschalten des Computers leicht überprüfen.

```
FOR X=139 TO 143:PRINT
PEEK(X):NEXT
```

Nach den Manipulationen des RND-Befehls wird das Resultat wieder in die Zellen 139 bis 143 als neuer Ausgangswert (seed) für den nächsten RND-Befehl gebracht.

Diese fünf Zahlen stellen eine Gleitkommazahl dar. Ihre Form entspricht dabei der Aufteilung, wie sie auch im Gleitkomma-Akkumulator (97 bis 101) verwendet wird.

Eine Abfrage dieser Zahlen aus den Zellen 139 bis 143 ist natürlich möglich, aber nicht ergiebig, weil das Resultat von RND(X) direkt als Zahl verfügbar ist, während die 5 Byte erst in eine brauchbare Zahl umgerechnet werden müßten. Eine Änderung durch POKEn neuer Werte in diese Speicherzellen geht leider nicht.

Adresse 144 ($90)

Statusvariable ST

Diese Adresse enthält ein Byte, welches mit der Statusvariablen ST von Basic identisch ist. Diese reservierte Variable ist im Texteinschub Nr. 14 »ST-atus« näher beschrieben.

Alle Routinen des Betriebssystems, die mit Ein- und Ausgabe zu tun haben, benutzen diese Speicherzelle zum Abspeichern und Abfragen des Status der Ein-/Ausgabeoperationen.

Genauer gesagt, alle Ein-/Ausgabeoperationen, die mit der Datasette und mit dem Floppy-Gerät beziehungsweise dem Drucker zu tun haben, benutzen die Adresse 144. Im Fachjargon sprechen wir vom Kassetten-Port und vom seriellen Port.

Der dritte Anschluß des Computers, nämlich der RS232 oder User-Port, benutzt für den Status die Speicherzelle 663.

Kassette:		
Bit 2	(Wert 4)	Kurzer Block
Bit 3	(Wert 8)	Langer Block
Bit 4	(Wert 16)	Lesefehler (nicht korrigierbar)
Bit 5	(Wert 32)	Prüfsummenfehler
Bit 6	(Wert 64)	File-Ende
Bit 7	(Wert 128)	Band-Ende
Floppy/Drucker:		
Bit 0	(Wert 1)	Fehler beim Schreiben
Bit 1	(Wert 2)	Fehler beim Lesen
Bit 6	(Wert 64)	Daten-Ende
Bit 7	(Wert 128)	»Device Not Present«-Fehler

Jedes Bit der Zelle 144 hat eine eigene Bedeutung wie folgt.

Alle nicht aufgeführten Bits sind nicht benutzt.

Diese Speicherzelle beziehungsweise die Statusvariable ST kann recht nützlich sein. Einige Kochrezepte dafür werden im Texteinschub Nr. 14 behandelt.

Adresse 145 ($91)

Zwischenspeicher für Abfrage der STOP-Taste

In den Bildern 13 und 14 ist dargestellt, wie die Tasten des VC 20 und des C 64 miteinander über eine Matrix verbunden sind.

60mal in der Sekunde unterbricht der Computer seine Arbeit, merkt sich, wo er gerade ist und fragt dann unter anderem, ob die STOP-Taste gedrückt worden ist. Dadurch wird erreicht, daß die STOP-Taste jederzeit Priorität hat.

Die Abfrage geht so vonstatten, daß das Betriebssystem über das im Bild 13 und 14 gezeigte Spaltenregister 56320 (beim VC 20: 37152) diejenige Tastenspalte anwählt, in welcher sich die STOP-Taste befindet. Aus Bild 13 und 14 sehen wir, daß dies die Spalte mit der Codenummer 127 beziehungsweise 247 ist. Ist in dieser Spalte eine Taste gedrückt, wird an ihrer Stelle eine Null in das Auslese-Register 56321 (VC 20: 37153) geschrieben. Die dadurch entstandene Dualzahl wird in die Speicherzelle 145 gebracht.

Es ist sicher verständlich, daß auf diese Weise nicht nur die STOP-Taste, sondern alle Tasten der Spalte 127 (247) abgefragt werden können. Ein kleines Demonstrationsprogramm kann das beweisen:

```
10 PRINT PEEK (656321);
PEEK (145)
20 GOTO 10
```

Beim VC 20 ist statt 56321 natürlich 37153 einzusetzen.

Das Zahlenband kann durch die Tasten der genannten Spalte – und nur durch diese – beeinflußt werden.

Adresse 146 ($92)

Zeitkonstante beim Lesen vom Band

Die Speicherzelle enthält eine vom Betriebssystem einstellbare Zahl, welche die kleinen Unterschiede in der Aufnahmegeschwindigkeit ausgleicht, die bei verschiedenen Datasetten vorkommen können.

Diese Zeitkonstante steht im Zusammenhang mit der Zahl, die in den Speicherzellen 176 und 177 steht.

Eine Veränderung der Konstante in Basic ist nicht möglich.

Adresse 147 ($93)

Flagge für LOAD oder VERIFY

Diese Flagge dient dem Betriebssystem, um zu unterscheiden, ob eine LOAD-Operation nur LOADen oder aber VERIFYen soll.

Sie ist identisch mit der Flagge des Basic-Übersetzers in Speicherzelle 10. Genauere Hinweise bitte ich der Beschreibung von Zelle 10 zu entnehmen.

Adresse 148 ($94)

Flagge für Floppy/Drucker-Ausgang

Das Betriebssystem benutzt diese Speicherzelle, um anzuzeigen, daß ein Zeichen im Ausgabepuffer steht, welches zum Floppy-Gerät oder zum Drucker geleitet werden soll. Diese Flagge setzt alle am seriellen Port angeschlossenen Geräte in den Zustand »Listen«, das heißt bereit zu sein, Daten aufzunehmen.

Adresse 149 ($95)

Zeichen im Ausgabepuffer

In dieser Speicherzelle wird das Zeichen abgelegt, welches als nächstes über den Serial-Port zum Floppy-Gerät oder zum Drucker transportiert wird, sobald die Flagge in 148 die Bereitschaft anzeigt.

Adresse 150 ($96)

Arbeitsspeicher für die Band-Leseroutine

Diese Speicherzelle wird zur Zwischenspeicherung von Daten beim Lesen einer Kassette benutzt.

Adresse 151 ($97)

Zwischenspeicher des X-Registers

Maschinen-Programmierer kennen das X-Register des Mikroprozessors. Beim Lesen eines Zeichens von der Datasette wird der Inhalt des X-Registers in dieser Adresse zwischengespeichert.

Adresse 152 ($98)

Anzahl der offenen Files

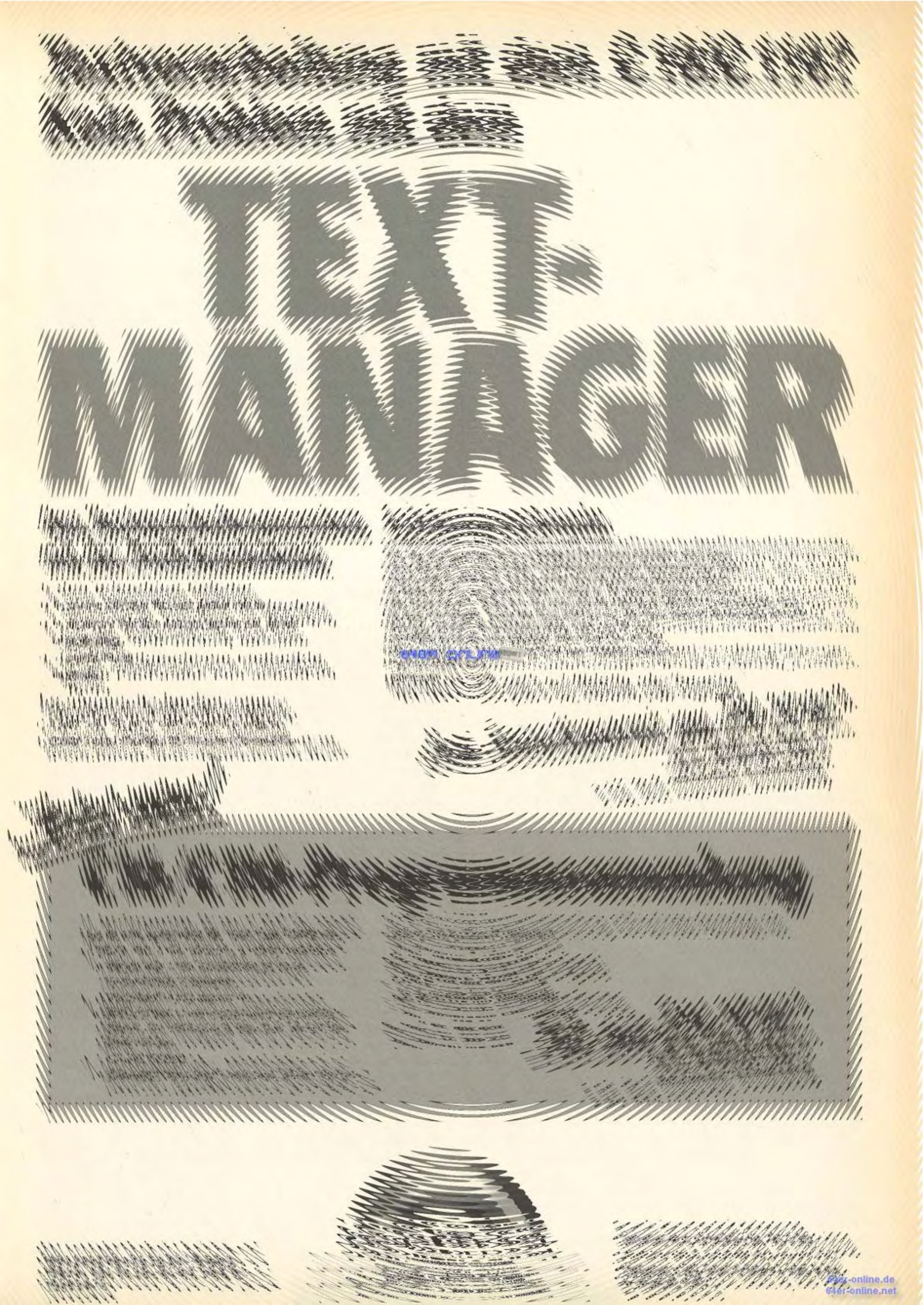
Ein File, oder auf Deutsch gesagt, eine Datei, wird mit dem Befehl OPEN eröffnet. Nach OPEN folgt die Nummer der Datei; sie ist beliebig wählbar bis maximal 255. Als zweites folgt die Nummer des Gerätes, mit dem die Verbindung hergestellt werden soll.

Es ist erlaubt, mehrere Dateien gleichzeitig geöffnet zu halten, vorausgesetzt die Nummern der Dateien sind verschieden.

In Speicherzelle 152 wird festgehalten, wieviel Dateien gleichzeitig geöffnet sind. Dieses kleine Programm zeigt es uns deutlich:

```
10 FOR K=10 TO 22
20 PRINT PEEK (152),K
30 OPEN K,0
40 NEXT K
```

Mit der FOR...NEXT-Schleife der Zeilen 10 und 40 eröffnen


TEXT-
MANAGER

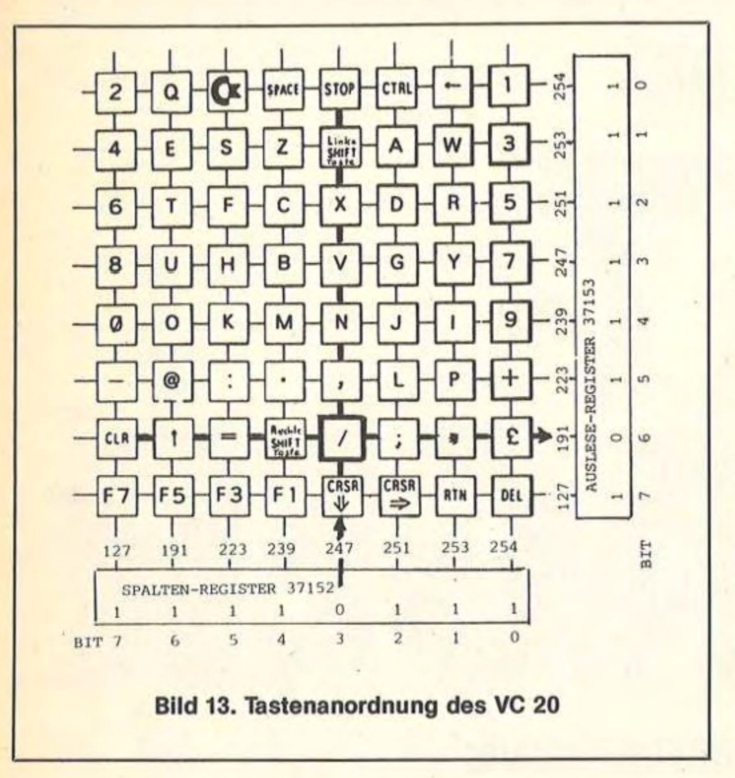


Bild 13. Tastenanordnung des VC 20

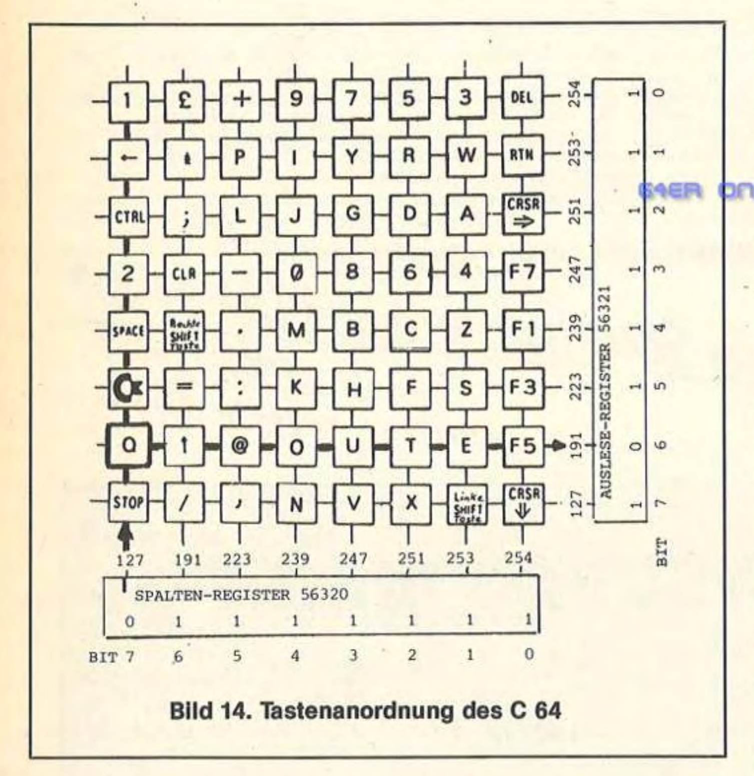


Bild 14. Tastenanordnung des C 64

wir 13 Dateien hintereinander, und zwar – wie Zeile 30 uns deutlich macht – mit der Tastatur. Die Tastatur hat die Nummer 0, der Drucker die Nummer 4, das Floppy-Gerät die Nummer 8 und die Datasette die Nummer 1. Ich habe die Tastatur gewählt, obwohl es keinen Sinn ergibt, weil sie die vielen Eröffnungen ohne zu unterbrechen akzeptiert.

Nach RUN sehen wir links untereinander den Inhalt von 152, also die Anzahl der eröffneten Dateien. Rechts steht jeweils die Nummer der eröffneten Datei.

Nach der 10. Datei bricht das Programm ab und druckt uns die Fehlermeldung TOO MANY FILES aus.

Das heißt es sind gleichzeitig nur 10 Dateien betreibbar. Wenn wir oben in Zeile 10 die Zahl 22 durch 19 ersetzen, läuft das Programm fehlerfrei.

Eine Datei, die unter einer bestimmten Nummer eröffnet ist, kann nicht noch einmal eröffnet werden. Fügen Sie bitte dem Programm noch die folgende Rücksprungzeile hinzu:

```
50 GOTO 10
```

In der 10. Zeile sehen wir jetzt die 10 als Inhalt von 152 und als neue Dateinummer (Schleifenvariable K) wieder die 10. Das Programm bleibt aber stehen und meldet FILE OPEN. Es hat recht, denn die Datei 10 ist bereits als erste eröffnet, aber nicht wieder geschlossen worden.

Das Betriebssystem macht das so, daß jede der Dateinummern in eine Tabelle geschrieben wird, die in den Speicherzellen 601 bis 610 stehen. Bei jedem OPEN-Befehl wird dort nachgeschaut, ob die Filenummer existiert. Wenn ja, wird die Fehlermeldung FILE OPEN ERROR ausgegeben. Bei jedem CLOSE-Befehl wird die entsprechende Nummer aus der Tabelle gelöscht.

Wir können aber auch eine 0 in die Speicherzelle 152 POKEn, wodurch dem Betriebssystem vorgegaukelt wird, daß keine Datei eröffnet ist. Schieben Sie im Programm einfach die Zeile ein:

```
45 POKE 152,0
```

und das Programm läuft ewig weiter.

Die Speicherzelle 152 ist also der Wächter über die Anzahl der eröffneten Dateien. Steht sie auf 0, dann wird eine Neueröffnung am Anfang der Tabelle ab 601 eingetragen. Die Tabelle ihrerseits ist der Wächter über Exklusivität der Dateinummern. Ich zeige Ihnen das noch genauer, wenn wir zu 601 kommen.

Sie werden vielleicht fragen, warum ich das so ausführlich beschreibe. Nun, in einem Programm kann es sicher sehr nützlich sein, die Zelle 152 mit PEEK nach der Datei-Lage abzufragen und entsprechend Maßnahmen zu treffen, ehe die Fehlermeldung das Programm abbricht.

Mit POKE 152,0 aber müssen Sie aufpassen. Es ersetzt nämlich nicht (!!) den CLOSE-Befehl. Probieren Sie es aus: Um das kleine Programm oben per Drucker auszudrucken, brauchen wir:

```
OPEN 1,4: CMD 1: LIST
```

Wenn Sie jetzt die Zeile 152 auf 0 POKEn und dann LIST eintippen, wird trotzdem wieder auf dem Drucker gelistet und nicht auf dem Bildschirm. Die vorgeschriebene Schließmethode mit

```
PRINT #1:CLOSE 1
```

geht jetzt aber auch nicht mehr, denn das Betriebssystem ist ja im Glauben, daß keine Datei eröffnet ist – schöner Schlamassel!

Erst eine Neueröffnung bringt alles wieder in die Reihe. Also Vorsicht mit der Anwendung der Speicherzelle 152. Eine Möglichkeit, alle Dateien auf einen Schlag zu schließen, gibt es aber doch.

SYS 65511 besorgt das sowohl beim C 64 als auch beim VC 20.

Adresse 153 ($99)

Nummer des Eingabe-Gerätes

Das Betriebssystem verwendet diese Speicherzelle, um festzuhalten, welches Gerät zur Eingabe verwendet werden soll.

Die Nummern sind wie folgt festgelegt:

```
0    = Tastatur
1    = Datasette
2    = RS232 (User-)Port
3    = Bildschirm
4, 5 = Drucker
8-11 = Floppy-Laufwerke
```

Nach dem Einschalten oder nach RESET des Computers steht in 153 eine 0 (Tastatur). Nach jedem Einsatz eines anderen Gerätes wird diese Speicherzelle wieder auf 0 gesetzt, so daß wir immer die Tastatur zur Verfügung haben.

Für Maschinenprogrammierer ist diese Adresse sicherlich wertvoll. Die Routine, welche die Eingabegeräte festlegt, sobald der Befehl INPUT# beziehungsweise GET# ausgeführt wird, heißt CHKIN und beginnt beim C 64 ab Adresse 61966 ($F20E), beim VC 20 ab 62151 ($F2C7).

Für Basic-Programmierer habe ich in der Literatur nur eine Anwendung gefunden, und die wurde bereits bei der Besprechung der Speicherzelle 19 angekündigt.

Es ist dies eine MERGE-Routine. Leider funktioniert dieses Verfahren nicht bei dem 1541-Floppy-Laufwerk. Erfunden wurde die Routine von Brad Templeton und ist von Jim Butterfield unter dem Namen »Magic Merge« für den VC 20/C 64 adaptiert worden. Ich gebe zu, in der Zwischenzeit sind noch andere, vielleicht auch kürzere MERGE-Routinen veröffentlicht worden. Aber diese hier verwendet gleich drei interessante Ingredienzen, nämlich die Speicherzellen 19 und 153 und außerdem die sogenannte »Dynamische Tastenabfrage«. Wer die letztere nicht kennt, sollte sich zum Verständnis den Texteinschub Nr. 15 gleichen Namens ansehen.

Ein MERGE (deutsch: zusammenführen, verschmelzen) be-

steht darin, ein auf Band gespeichertes Programm zu einem im Computer stehenden anderen Programm so dazuzuladen, daß dieses nicht überschrieben, sondern ergänzt wird. Wichtig ist dabei, daß das Programm vom Band höhere Zeilennummern hat als das Programm im Computer. Außerdem muß das Programm auf dem Band als Datei gespeichert sein. Das wird so erreicht:

1. Programm eintippen
2. Direkt eingeben:
   OPEN 1,1,1,"Name":
   CMD1:LIST
3. Erst wenn READY kommt, direkt eingeben
   PRINT#1:CLOSE1

Damit ist das Programm auf dem Band gespeichert. Nun kommt der eigentliche MERGE-Vorgang.

4. Es steht ein Programm im Computer
5. Band mit dem Programm »Name« einlegen
6. Direkt eingeben:
   POKE 19,1:OPEN 1
7. Sobald READY erscheint, Bildschirm löschen (SHIFT-CLR).
8. Dreimal Cursor-Down
9. Direkt eingeben:
   PRINT CHR$(19):
   POKE 198,1:
   POKE 631,13:
   POKE 153,1
10. Das Band beendet den Ladevorgang mit einer Fehlermeldung, die wir ignorieren.
11. Nach CLOSE 1 sind beide Programme zusammengefügt.

Wie gesagt, Schritt 6 verwendet Zelle 19 (bitte dort nachlesen), Schritte 8 und 9 sind die »Dynamische Tastenabfrage«, und Schritt 9 verwendet zusätzlich die hier zur Diskussion stehende Speicherzelle 153, um die Datasette als Eingabegerät zu definieren.

Adresse 154 ($9A)

Nummer des Ausgabe-Gerätes

Diese Speicherzelle entspricht der Zelle 153, nur steht hier die Nummer des Gerätes, über das die Ausgabe läuft.

Nach dem Einschalten und nach Ausgabeoperationen wird der Wert immer auf 3 gesetzt. Das ist entsprechend der oben genannten Zuordnung der Bildschirm.

Für Maschinenprogrammierer sei erwähnt, daß Basic bei den Befehlen PRINT# oder CMD die Routine CHKOUT einsetzt, welche die Adresse 154 belegt. Sie steht im C 64 ab Adresse 62032 ($F250), im VC 20 ab 62217 ($F309).

Adresse 155 ($9B)

Fehlerkontrolle bei Bandoperationen

Die Commodore-Datasette ist deswegen so zuverlässig, weil sie mehrere Methoden zur Fehlererkennung beziehungsweise Korrektur von Lese- und Schreibfehlern verwendet.

Eine der Methoden ist die sogenannte Parity-Prüfung. Sie ist nichts anderes als eine Quersummenbildung der einzelnen Stellen jedes Bytes, deren Resultat überprüft wird.

Die Speicherzelle 155 wird bei dieser Parity-Prüfung eingesetzt.

Adresse 156 ($9C)

Flagge für korrektes Byte vom Band

In dieser Speicherzelle wird zwischengespeichert, ob das vom Band gelesene Byte die Prüfungen bestanden hat, also richtig ist oder nicht.

Adresse 157 ($9D)

Flagge für Meldungen

Man muß zwischen zwei Arten von Meldungen unterscheiden:

Meldungen des Betriebssystems
Meldungen des Basic-Übersetzers

Die Meldungen des Betriebssystems kennen wir als Angaben zum Ablauf, wie SEARCHING FOR, FOUND, PRESS PLAY ON TAPE und so weiter. Normalerweise nicht bekannt ist die Meldung I/O ERROR #, wobei nach dem Zeichen # Zahlen von 0 bis 29 stehen können. Diese Zahlen beziehen sich auf Meldungen des Übersetzers (Interpreter), die ausschließlich Fehlermeldungen sind. Das mag verwirrend klingen, klärt sich aber sofort. Die Flagge in 157 kann vier Werte annehmen: 0, 64, 128 und 192.

1. Der Wert 0 unterdrückt alle Meldungen des Betriebssystems. Dieser Modus tritt nach RUN beim Ablauf eines Programms ein.
2. Der Wert 64 läßt nur Fehlermeldungen des Betriebssystems zu. Dieser Modus ist normalerweise nicht vorgesehen, kann aber künstlich erzeugt werden.
3. Der Wert 128 unterdrückt die Fehlermeldung des Betriebssystems. Dieser Modus entspricht dem Normalfall.
4. Der Wert 192 läßt alle Meldungen zu. Auch dieser Modus ist nur künstlich herzustellen.

Das folgende Beispiel macht das deutlich. Geben Sie direkt ein:

POKE 157,0:LOAD "$",9

Wir versuchen, vom Gerät mit der Nummer 9, das ist eine zweite Floppy, die Directory zu laden. Wir erhalten entsprechend Punkt 1 nur die Meldung des Übersetzers »? DEVICE NOT PRESENT«.

Verändern wir den POKE-Befehl für Punkt 2:

POKE 157,64:LOAD"$",9

Wir erhalten jetzt »I/O ERROR #5 ? DEVICE NOT PRESENT«.

POKE 157,128:LOAD"$",9

ergibt die Meldung »SEARCHING FOR $
? DEVICE NOT PRESENT«.

Schließlich nehmen wir noch den letzten Fall:

POKE 157,192:LOAD"$",9

Jetzt erhalten wir alles:
»SEARCHING FOR $
I/O ERROR #5
? DEVICE NOT PRESENT«

Da die Fehlermeldung des Betriebssystems und die zugehörigen Nummern in keinem Handbuch erwähnt sind, habe ich sie interessehalber in der folgenden Tabelle zusammengefaßt.

#	MELDUNG (ERROR)
1	TOO MANY FILES
2	FILE OPEN
3	FILE NOT OPEN
4	FILE NOT FOUND
5	DEVICE NOT PRESENT
6	NOT INPUT FILE
7	NOT OUTPUT FILE
8	MISSING FILE NAME
9	ILLEGAL DEVICE NUMBER
10	NEXT WITHOUT FOR
11	SYNTAX
12	RETURN WITHOUT GOSUB
13	OUT OF DATA
14	ILLEGAL QUANTITY
15	OVERFLOW
16	OUT OF MEMORY
17	UNDEF'D STATEMENT
18	BAD SUBSCRIPT
19	REDIM'D ARRAY
20	DIVISION BY ZERO
21	ILLEGAL DIRECT
22	TYPE MISMATCH
23	STRING TOO LONG
24	FILE DATA
25	FORMULA TOO COMPLEX
26	CAN'T CONTINUE
27	UNDEF'D FUNCTION
28	VERIFY
29	LOAD

Texteinschub Nr. 14
ST-atus

Neben den Befehlen (wie PRINT) und den Funktionen (wie COS) hat Basic auch noch drei fest definierte Variable, nämlich TI, TI$ und ST.

Von den dreien ist ST wohl am seltensten anzutreffen, Grund genug, hier ein wenig darüber zu berichten.

Der Anlaß ist natürlich, daß der Wert von ST immer in der Speicherzelle 144 steht, die ja hier vorkommt.

Bei der Beschreibung wurde schon erwähnt, daß ST den Status nach der letzten Ein- beziehungsweise Ausgabeoperation angibt, beschränkt allerdings nur auf Operationen mit der Datasette und der an einem gemeinsamen Ausgang angeschlossenen Floppy und Drucker.

Dementsprechend zeigt die Tabelle bei der Speicherzelle 144 diese beiden Fälle.

Wichtig ist noch zu erwähnen, daß nicht nur die in der Tabelle gezeigten Zahlen für ST auftreten, sondern auch Kombinationen davon. So ergibt zum Beispiel ein zu kurzer Block (4) und ein gleichzeitig aufgetretener Prüfsummenfehler (32) einen Wert von 36.

Kassettenoperationen

Zuerst testen wir mit einem Datei-Programm auf »File-Ende«. Geben Sie bitte folgendes Programm ein:

```
10  OPEN 1,1,1,"DATEI"
20  PRINT#1, "QWERTY"
30  CLOSE 1
40  END
```

Zur Erinnerung: Nach dem OPEN-Befehl folgt zuerst die Nummer der Datei (ich nehme hier 1), dann die Gerätenummer (1= Datasette) und schließlich die Sekundäradresse (1= Schreiben).

Jetzt kommt der Lesevorgang:

```
 50 OPEN 2,1,0"DATEI"
 60 FOR K=1 TO 10
 70 GET#2,A$
 80 PRINT A$;ST
 90 NEXT K
100 CLOSE 2
```

In Zeile 50 eröffnen wir wieder eine Datei (diesmal Nummer 2) für die Datasette, jetzt aber zum Lesen (Sekundäradresse=0). Die Schleifen der Zeilen 60 und 90 schreiben uns 10mal ein Zeichen (A$) und den Wert von ST auf den Bildschirm.

Jetzt geht es los. Mit RUN starten wir den ersten Teil des Programms. Nach dem Schreibvorgang und der READY-Meldung (nach Zeile 40) müssen Sie das Band zurückspulen und mit GOTO 50 ab Zeile 50 weiterfahren. Jetzt wird die Datei gelesen.

Wir erhalten untereinander die sechs Buchstaben von Zeile 20, daneben für ST lauter Nullen. Am Ende allerdings erscheint eine 64. Das ist der in der Tabelle angegebene Wert von ST für »File-Ende«.

Da die FOR-NEXT-Schleife zu lang ist, schießen wir mit dem Lesen über das File-Ende hinaus. Normalerweise kennen wir natürlich die Länge einer Datei nicht. Deshalb ist es besser, mit GOTO zurückzuspringen und das File-Ende abzufragen.

Löschen Sie bitte Zeile 60 und 90 und fügen Sie als Rücksprung und Prüfung ein:

```
85  IF ST=64 THEN 100
90  GOTO 70
```

Statt nach ST können wir natürlich genausogut nach PEEK(144) fragen.

Ein erneutes GOTO 50 bringt das erwünschte Resultat.

Um den vorhin schon erwähnten »kurzen Block« zu sehen, müssen wir einen entsprechenden Fehler künstlich erzeugen.

Löschen Sie bitte den ersten Teil des Programms bis einschließlich Zeile 40. Wir behalten also nur den Leseteil ab Zeile 50. Dann laden wir dieses Programm (Band vorher am besten wieder zurückspulen) mit SAVE "DATEI" nicht als Datei, sondern als ganz gewöhnliches Programm. Wenn es geladen ist, bitte das Band wieder zurückspulen.

Mit RUN starten wir jetzt das Lese-Programm, welches eine Datei sucht, aber nur ein Programm findet, allerdings mit dem richtigen Namen. Natürlich findet es einen Fehler und wir erhalten als Ausdruck:

36 oder manchmal auch 4
64 64

Die Zahl 36 entsteht aus 32+4, das bedeutet Prüfsummenfehler + Block zu kurz. Danach folgt wie vorher das File-Ende.

Die normale Blocklänge entspricht der Länge des Bandpuffers, in den die Datasette einspeichert. Er ist 191 Byte lang. In unserem Fall war offenbar der Block nicht ganz voll.

Der Prüfsummenfehler tritt dann auf, wenn eine der Überprüfungen von Kassettenoperationen einen Fehler gefunden hat. Der Blockfehler, auch der des zu langen Blocks, interessiert wohl weniger. Aber ein durch die Prüfungen gefundener Fehler könnte frühzeitig, noch vor dem Ausstieg des Programms, entdeckt, abgefangen und ausgenutzt werden.

Die Formel dafür, ins obige Programm eingebaut, ist:

85 IF ST<64 OR ST>8 THEN..(zum Beispiel LIST)

Statt LIST kann man natürlich auch etwas anderes nehmen.

Floppy-Operationen

Bei der Floppy bedeutet ST=64, »Daten-Ende«, das ist etwa dasselbe wie bei der Datasette. Um es zu überprüfen, nehmen wir dasselbe Programm wie vorher, nur müssen wir die Datei-Zeilen der Floppy anpassen. Das sieht dann so aus:

```
 10 OPEN1,8,1"DATEI,S,W"
 20 PRINT#1,"QWERTY"
 30 CLOSE 1
 40 END
 50 OPEN 2,8,0"DATEI,S,R"
 60 FOR K=1 TO 10
 70 GET#2,A$
 80 PRINT A$,ST
 90 NEXT K
100 CLOSE 2
```

Das Ergebnis sieht hier so aus:

64
66
66

Die 64 ist natürlich wie erwartet der »Wert« für Daten-Ende. Die 66 ist 64 + 2, entstanden dadurch, daß wir über das Daten-Ende hinausgelesen haben. Die 2 bedeutet »Fehler beim Lesen« (in den englischen Beschreibungen heißt es »Read Time Out«). Ähnliches gilt für ST=1, das bedeutet »Fehler beim Schreiben« (englisch: Write Time Out), nur weiß ich leider nicht, wie es vorzuführen ist. Wie bei der Datasette kann das Überlesen natürlich mit der Abfrage IF ST=64 THEN...und GOTO...gestoppt werden.

Interessant ist noch der Status beim Fehler »DEVICE NOT PRESENT«, den wir dadurch erzeugen, daß wir ein Programm oder die Directory von der Diskette laden wollen, ohne daß dieses Gerät angeschlossen oder eingeschaltet ist. Nach der Fehlermeldung geben wir direkt ein:

PRINT ST oder PRINT PEEK(144)

und wir erhalten die Zahl 128.

Wie man allerdings in einem Basic-Programm durch Abfrage von ST=128 die Fehlermeldung »Device Not Present« und den dann folgenden Programmabbruch vermeiden kann, bedarf einer gesonderten Maßnahme:

Es gibt zwei Speicherzellen – 768 und 769 –, auf die wir bei unserer Wanderung durch die Speicherlandschaft noch kommen werden, in denen in Low-/High-Byte-Darstellung eine Adresse steht, auf die das Betriebssystem springt, wenn die Meldung »DEVICE NOT PRESENT« ausgegeben werden soll. Diesen Zeiger kann man so »verbiegen«, daß die Meldung nicht ausgegeben wird und daß das Programm einfach weiterläuft.

Normalerweise steht in 768 die Zahl 139 (VC 20: 58), in 769 die Zahl 227.

Verbogen wird der Zeiger durch eine 61 (185 geht auch), beim VC 20 durch 52. Dadurch zeigt die Adresse auf eine Speicherzelle des Betriebssystems, in welcher der Assembler-Befehl »RTS«, das bedeutet Rücksprung, steht. Jetzt können wir ungestört den STatus abfragen, wir müssen allerdings den negativen Wert von ST, also -128 nehmen. Das geänderte Programm sieht dann so aus:

```
 10 POKE 768,61                 Fehlermeldung abschalten
 20 OPEN 1,8,15                 Gerät ansprechen
 40 CLOSE 1
 50 POKE 768,139                Fehlermeldung einschalten
 60 IF ST= -128 THEN 100        Sprung bei ausgeschaltetem
                                Gerät
 70 PRINT »FORTSETZUNG«         weiter im Programm
 80 END                         Ende des DEMO
100 PRINT »GEREAT EINSCHALTEN«
110 GET A$ : IF A$ = "" THEN 10 Warteschleifen

120 GOTO 10                     neuer Versuch
```

Drucker-Operationen

Für den Drucker sieht die Änderung fast genauso aus. Die einzige Änderung ist in den Zeilen 20 und 30

```
20 OPEN 1,4
30 PRINT#1
```

Unter bestimmten Umständen kann das Verbiegen des Zeigers in 768 entfallen.

Ich möchte nach eigenen längeren Versuchen aber dafür plädieren, die Fehlermeldung immer abzuschalten, um nie in Schwierigkeiten zu kommen. Vorsicht ist die Mutter der Weisheit.

Texteinschub Nr. 15
Dynamische Tastenabfrage

Jedesmal, wenn Sie eine Taste drücken, wird der ASCII-Codewert des Zeichens oder der Funktion dieser Taste ermittelt und im »Tastaturpuffer« gespeichert. Dieser Pufferspeicher liegt in den Speicherzellen 631 bis 640.

Die eigentliche Abfrage und Umcodierung eines Tastendrucks habe ich im Kurs »Alle Tasten-, Zeichen- und Steuercodes« in den Ausgaben 4/84 bis 7/84 beschrieben. Er ist auch im 64er-Sonderheft Nr. 2/86 enthalten. Die ASCII-Codetabelle finden Sie da auch. Sie ist natürlich in allen Handbüchern enthalten, leider aber nicht immer ganz vollständig.

Zuerst will ich Ihnen die Arbeitsweise des Tastaturpuffers zeigen.

Der Computer holt sich den ASCII-Codewert aus dem Tastaturpuffer und wenn gerade kein Programm läuft, druckt er das Zeichen auf den Bildschirm oder führt die Funktion der Taste aus. Das ist der oft zitierte »Direkt-Modus«.

Wenn aber ein Programm läuft, dann bleiben die Codezahlen im Puffer so lange stehen, bis der Computer fertig ist. Dann erst werden sie herausgeholt und verarbeitet. Das will ich Ihnen beweisen.

Tippen Sie im Direkt-Modus ein:

```
FOR K=1 TO 15000:NEXT K (RETURN)
```

Während diese an sich sinnlose Zeitschleife 15000mal im Kreis rumrennt, haben Sie genügend Zeit, mehrere Tasten zu drücken, zum Beispiel die erste Buchstabenreihe (QWERTYUIOP@*↑). Natürlich sehen Sie am Bildschirm gar nichts, denn das Programm der Schleife läuft ja noch. Sobald aber die Schleife zu Ende ist, erscheinen 10 der getippten Buchstaben. Quod erat demonstrandum!

Warum nur 10 Buchstaben? Nun, der Tastaturpuffer hat nur 10 Plätze, logisch?

Jetzt müssen wir noch eine weitere Speicherzelle ins Spiel bringen. In die Zelle 198 kann man nämlich eine Zahl hineinPOKEn, welche die Anzahl der Zeichen im Tastaturpuffer begrenzt.

Wiederholen Sie bitte das Experiment von oben, nur soll die direkt eingegebene Zeile erweitert werden:

```
FOR K=1 TO 15000:NEXT K: POKE 198,6 (RETURN)
```

Und siehe da, jetzt werden nur die sechs Buchstaben Q bis Y ausgedruckt. Diese Anwendung des Tastaturpuffers nutzen wir für das Kochrezept »Dynamische Tastenabfrage« aus. Allerdings müssen wir dazu prinzipiell die ASCII-Codewerte verwenden, so wie im nächsten Beispiel.

Löschen Sie bitte den Bildschirm und geben Sie ein (identisch für VC 20 und C 64):

```
10 PRINT CHR$(65)
20 PRINT CHR$(156)
30 PRINT CHR$(66)CHR$(13)CHR$(67)
```

65 ist der Code für A, 156 für die Farbe »purple«, 66 für B, 13 für »RETURN« und 67 für C. Das heißt also, daß wir zuerst ein A drucken, dann auf »purple« umschalten, darunter dann das B schreiben, einmal RETURN geben und dann noch das C folgen lassen.

Bild 15 zeigt den Ausdruck auf dem Bildschirm, wenn Sie diese Zeilen LISTen und dann mit RUN starten.

Zur Erklärung: Die Leerzeile zwischen A und B ist bedingt durch die PRINT-Anweisung in Zeile 20, welche nur die Farbe umschaltet. Obwohl die Codes für B und C zusammen in einer Zeile stehen, werden sie doch durch das »RETURN« CHR$(13) untereinandergesetzt.

Anstelle der 13 können Sie alle möglichen anderen Steuerfunktionen setzen. Bild 16 zeigt das Resultat von 17, nämlich »Cursor down«. Wenn Sie die 8 nehmen, können Sie den Zeichensatz nicht mehr ändern. Der Einsatz der gleichzeitig gedrückten SHIFT- und Commodore-Taste funktioniert erst nach CHR$(9) wieder.

Als nächstes wollen wir die ASCII-Codewerte und den Tastaturpuffer im »Programm-Modus« einsetzen.

Das Resultat von Bild 16 wollen wir jetzt durch POKEn der ASCII-Werte in den Tastaturpuffer wiederholen.

```
5  POKE 198,5
10 POKE 631,65
20 POKE 632,156
30 POKE 633,66: POKE 634,17: POKE 635,67
99 END
```

Prinzipiell macht dieses Programm das gleiche wie das Programm in Bild 16. Trotzdem erhalten wir nach LIST und RUN ein anderes Ergebnis, nämlich das von Bild 17.

Ist das ein Fehler? Natürlich nicht. Nach RUN laufen zuerst einmal alle POKE-Befehle ab. Zeile 5 gibt an, wieviele Zeichen im Puffer stehen sollen. In Zeile 99 findet das Programm das ENDe und meldet sich mit READY. Jetzt erst wird im Tastaturpuffer nachgeschaut. Dort findet der Computer zuerst das A, dann »purpur«, dann das B, welches sofort neben das A gesetzt wird. Das ist auch logisch, denn es fehlt ja jede Angabe, eine Zeile tiefer zu gehen. Um das zu erreichen, müssen wir in der Zeile 10 den Codewert für RETURN einschieben:

```
10 POKE 631,65: POKE 632,13
```

Vorsicht! Sie müssen in Zeile 20 und 30 alle POKE-Adressen um 1 erhöhen und auch die Zahl in Zeile 5. Nehmen Sie 10, dann haben Sie Platz für Erweiterungen. So, jetzt LIST und RUN eingeben und es erscheint Bild 18 – und wir haben schon wieder ein Problem! Aber alles im Computer ist logisch! Nach dem A findet er den Wert für »RETURN«, also führt er den Befehl aus, auf dem der Cursor gerade steht. Der steht auf dem A. Da das kein gültiger Basic-Befehl ist, druckt der Computer die Fehler-Meldung und zeigt READY an. Danach allerdings macht er weiter wie oben.

Und jetzt kommt die ASCII-Zahl 141 (SHIFT-RETURN) voll zur Geltung. Diese Kombination nämlich setzt den Cursor an den Anfang der nächsten Zeile, ohne »RETURN« auszuführen.

Ersetzen Sie also die 13 in Zeile 10 durch 141, dann läuft's (Bild 19).

Es gibt übrigens noch eine interessante ASCII-Codezahl, die in keiner Tabelle steht, nämlich 131. Sie bedeutet dasselbe wie die geSHIFTete STOP-Taste, also die Funktion »LOAD+RUN«.

Wenn Sie diesen Code mit PRINT CHR$(131) ausprobieren, funktioniert er allerdings nicht. Deshalb steht er wohl auch nicht in den Tabellen.

In den Tastaturpuffer gePOKEt bringt er aber seine Wirkung.

Setzen Sie bitte in Zeile 30 an die Stelle von 67 die Zahl 131 und anstelle des C erscheint:

LOAD und PRESS PLAY ON TAPE. Gut, nicht wahr?

So, jetzt haben wir alle Zutaten für unser Kochrezept zusammen.

Löschen Sie bitte alles bisherige und tippen Sie ein:

```
10 PRINT CHR$(147)
20 FOR I=1 TO 5: PRINT I
30 FOR T=1 TO 500: NEXT T
40 NEXT I
```

Nach Löschen des Bildschirms (Zeile 10) drucken wir zum Ausschmücken die Zahlen 1 bis 5 untereinander (Zeile 20 und 40) und damit es nicht zu schnell geht, bremsen wir mit der Zeile 30.

```
50 PRINT "LIST" (Das ist natürlich sehr einfach, aber
   jetzt kommt's!)
60 POKE 198,5
70 POKE 631,145: POKE 632,145: POKE 633,145
80 POKE 634,13
90 END
```

Nach RUN erscheinen erst die fünf Zahlen, dann wird in einer neuen Zeile das Wort LIST geschrieben. Nach END wird erst (wie immer) eine Zeile ausgelassen, dann READY gedruckt und schließlich springt der Cursor an den Anfang der Zeile darunter. Während der Cursor anfangen will, drei Zeilen unter dem Wort LIST zu blinken, findet der Computer im Tastaturpuffer 3 den ASCII-Code für »Cursor up«. Also geht dieser auch drei Zeilen hoch und will jetzt auf dem Wort LIST blinken.

Damit wird es aber wieder nichts, denn im Puffer steht ja noch

der Code für RETURN (13). Das wird ausgeführt und zwar für das LIST. Es hat dieselbe Wirkung, als ob Sie direkt LIST tippen und danach auf die RETURN-Taste drücken, nämlich das Programm wird ausgeLISTet.

Alle sinnvollen Basic-Befehle, die Sie in der Zeile 50 PRINTen, werden durch diese dynamische Manipulation des Cursors ausgeführt.

Versuchen Sie es mit

```
50 PRINT →PRINT 16 * 35"
50 PRINT "LOAD"
50 PRINT "GOTO 10"
50 PRINT "RUN"
50 PRINT "RUN 50"
```

und falls Sie dieses kleine Programm geSAVEt haben

```
50 PRINT "SYS 64824"
50 PRINT "SYS 64763"
```

Die Kunst ist also, mit entsprechenden Codezahlen den Cursor an diejenige Stelle des Bildschirms zu bringen, wo innerhalb eines Programms eine geeignete Anweisung gedruckt worden ist. Man kann damit getrennte Programmteile nachladen, mit SYS-Befehlen Maschinenprogramme aufrufen, oder gar Programme durch sich selbst ändern lassen.

```
LIST
10 PRINT CHR$(65)
20 PRINT CHR$(156)
30 PRINT CHR$(66) CHR$
 (13) CHR$(67)
READY.

RUN
A

B
C
READY.
```

Bild 15. Testprogramm

```
LIST
10 PRINT CHR$(65)
20 PRINT CHR$(156)
30 PRINT CHR$(66) CHR$
 (17) CHR$(67)
READY.

RUN
A
B
 C
READY.
```

Bild 16. CR ersetzt durch »Cursor down«

```
LIST
5 POKE 198,5
10 POKE 631,65
20 POKE 632,156
30 POKE 633,66:POKE 63
4,17:POKE 635,67
40 END
READY.

RUN
READY.
AB
 C
```

Bild 17. Programm (Bild 16) in den Tastaturpuffer gePOKEt

```
5 POKE 198,10
10 POKE 631,65:POKE 63
2,13
20 POKE 633,156
30 POKE 634,66:POKE 63
5,17:POKE 636,67
40 END
READY.
RUN
READY.
A
?SYNTAX
 ERROR
READY.
B
 C
```

Bild 18. Nach »A« wird CR eingefügt

```
LIST
5 POKE 198,10
10 POKE 631,65:POKE 63
2,141
20 POKE 633,156
30 POKE 634,66:POKE 63
5,17:POKE 636,67
40 END
READY.

RUN
READY.
A
B
 C
```

Bild 19. SHIFT-CR statt CR

Adresse 158 und 159 ($9E und $9F)

Zwischenspeicher bei Kassettenoperationen

Diese beiden Speicherzellen werden von Routinen des Betriebssystems verwendet, welche bei Kassettenoperationen die Zeichen überprüfen, ob sie richtig sind, und welche bei aufgetretenen Fehlern Korrekturen durchführen.

Adresse 160 bis 162 ($A0 bis $A2)

Interne Uhr für TI und TI$

Das Basic der Commodore-Computer kennt neben der Variablen ST (siehe Speicherzelle 144) noch zwei weitere »reservierte« Variable, nämlich TI und TI$. Beide bieten eine interne Uhr, welche aus dem Inhalt der Speicherzellen 160 bis 162 abgeleitet wird. Diese drei Zellen funktionieren wie der Kilometerzähler eines Autos, halt nur mit drei Stellen.

Die hinterste Stelle ist die Zelle 162. Ihr Inhalt wird beim Einschalten des Computers auf 0 gesetzt, dann aber 60mal in der Sekunde um 1 erhöht. Das erfolgt durch die automatische Interrupt-Routine, welche auch die STOP-Taste abfragt und noch andere Hausaufgaben 60mal in der Sekunde ausführt. Da $\frac{1}{60}$ = 0,01667 ist, zählt also die Zelle 162 in 0,01667 Sekunden um 1 weiter. Sie kann wie alle Speicherzellen maximal nur die Zahl 255 enthalten, danach kommt wieder eine 0. Das heißt aber, daß sie nach 256 * 0,01667 = 4,267 Sekunden einmal durchgelaufen ist.

Nach jedem Durchlauf wird die davorliegende Speicherzelle 161 um 1 erhöht. Sie zählt also in 4,267 Sekunden um 1 weiter und ist nach 256 * 4,067 = 1 092,26 Sekunden oder besser nach 18,2044 Minuten einmal durchgelaufen. Nach dem Kilometerzähler-Prinzip wird nach jedem Durchlauf von 161 der Inhalt der davorliegenden Zelle 160 um 1 erhöht.

Die Zelle 160 zählt also in 18,2044 Minuten um 1 weiter und ist nach 256 * 18, 2044 = 4660,34 Minuten, das sind 77,67 Stunden, einmal durchgelaufen.

Diese Stundenzahl wird allerdings niemals erreicht, da das Betriebssystem nach Erreichen des Wertes für 24 Stunden alle drei Zellen wieder auf 0 zurückgesetzt. Wir werden das gleich nachprüfen.

Zuerst aber wollen wir uns den dreizeiligen Zähler anschauen:

```
10 PRINT PEEK(160);
PEEK(161);PEEK(162)
20 GOTO 10
```

Nach RUN sehen wir den Inhalt der drei Zellen sich entsprechend der oben angegebenen Zeiten verändern. Die Zahlen sind nicht vorherbestimmbar, denn der Zähler ist ja nach dem Einschalten des Computers schon losgelaufen. Er kann aber auf 0 gesetzt werden durch Einfügen der Zeile 5:

```
5 POKE 160,0:POKE 161,0:
POKE 162,0
```

Jetzt beginnt der Zähler immer ab 0. Ich habe gerade gesagt, daß der Zähler auf 0 gesetzt wird, wenn er 24 Stunden lang gelaufen ist. Der Inhalt in den drei Speicherzellen, der 24 Stunden entspricht, ist nach der oben angegebenen Umrechnungsart 79-26-0. Diesen Wert, oder besser noch ein Wert kurz davor, in die Zellen 160 bis 162 gePOKEt, zeigt uns den Nullsetzvorgang. Ersetzen Sie bitte die obige Zeile 5 durch eine neue Zeile:

```
5 POKE 160,79:POKE 161,25:
POKE 162,0
```

Nach vier Sekunden Laufzeit schalten alle drei Zellen in der Tat auf 0 zurück.

Die Umsetzung der Zahlen aus 160 bis 162 in die Variablen TI und TI$ sowie deren Wirkungsweise entnehmen Sie bitte dem Texteinschub Nr. 16 »Die eingebaute Uhr«.

Abschließend muß eines noch warnend erwähnt werden. Alle Operationen, welche den Interrupt-Vektor verwenden beziehungsweise verändern, stören oder verzögern die normale Interrupt-Routine, die ja den Zähler weiterstellt. So zählt der Zähler nicht gleichmäßig und die daraus abgeleitete Uhr geht nicht mehr richtig. Ein Beispiel dafür sind alle Ein- und Ausgaben über die Datasette, welche über einen Interrupt laufen.

Adresse 163 und 164 ($A3 und $A4)

Zwischenspeicher

Diese beiden Speicherzellen werden von den Ein- und Ausgabe-Routinen des Betriebssystems für Kassetten, Floppy-Laufwerk und Drucker als Zwischenspeicher für alle möglichen Werte benutzt.

Adresse 165 ($A5)

Bit-Zähler für Kassetten-Synchronisierung

Beim Abspeichern eines Programms auf ein Band werden vor den eigentlichen Daten mehrere Bits zusätzlich gespeichert, die beim Einlesen dieses Bandes zur Synchronisierung dienen, das heißt zum Übereinstimmen der Geschwindigkeit der Datenübertragung.

Die Speicherzelle 165 wird als Zähler dieses Synchron-Bits verwendet.

Adresse 166 ($A6)

Zähler der bearbeiteten Bytes im Kassetten-Puffer

Diese Speicherzelle wird als Zähler benutzt, welcher angibt, wieviele Bytes gerade in den Kassetten-Puffer eingeschrieben oder aus ihm ausgelesen worden sind. Der Kassetten-Puffer besteht aus den Speicherzellen 828 bis 1 019 und kann somit 191 Byte aufnehmen, was zugleich die höchste Zahl ist, welche sinnvollerweise in der Zelle 166 stehen kann.

Nähere Erklärungen und ein paar Experimente mit Zelle 166 finden Sie in dem Texteinschub 17 »Experimente mit dem Kassetten-Puffer«.

Die meisten der nächsten 20 Speicherzellen werden bei Operationen mit der RS232-Schnittstelle, die über den User-Port den Computer mit anderen Geräten verbindet, eingesetzt. Da die Programmierung der RS232-Schnittstelle noch andere Speicherzellen benötigt, die später an der Reihe sind, gehe ich auf die RS232-Schnittstelle erst bei der Behandlung der Speicherzelle 659 bis 673 näher ein.

Adresse 167 ($A7)

Zwischenspeicher für Kassetten-Operationen und für Eingabe über die RS232-Schnittstelle

Diese Speicherzelle wird verwendet, um jedes Bit, welches von einem RS232-Kanal über den User-Port eingelesen wird, zwischenzuspeichern.

Außerdem verwenden mehrere Kassetten-Routinen diese Adresse als Zwischenspeicher.

Adresse 168 ($A8)

Bitzähler für RS232-Eingabe und bei Band-Ein-/Ausgabe

Die Speicherzelle 168 wird als Zähler verwendet, der diesmal nicht die Bytes, sondern die Anzahl der Bits zählt, die sowohl über den User-Port als auch über den Kassetten-Port geleitet werden. Das dient dem Betriebssystem dazu, zu wissen, wann ein volles Wort abgearbeitet worden ist.

Adresse 169 ($A9)

RS232-Flagge für Startbit-Prüfung

Ein RS232-Datentransfer prüft, ob ein Start-Bit empfangen worden ist. Im positiven Fall steht in Zelle 169 die Zahl 144, im negativen Fall eine 0.

Adresse 170 ($AA)

RS232-Eingabe- und Zwischenspeicher für Kassetten-Routinen

Bei der Speicherzelle 165 haben wir gesehen, daß ein Band Synchronisationsbits enthält. Die Speicherzelle 170 wird dabei als Flagge benutzt, die angibt, ob ein gelesenes Zeichen Synchronisierungs-Bits oder ein Datenwort darstellt.

Die RS232-Routinen verwenden Zelle 170 dagegen als Speicher, in welchem die eingelesenen Bits zu einem Byte zusammengefaßt werden, bevor sie im Eingabepuffer am oberen Ende des Programmspeichers abgelegt werden (siehe auch Speicherzellen 55/56).

Adresse 171 ($AB)

Quersummenprüfung und Zähler für Band-Header bei RS232- und Kassetten-Operationen

Diese Speicherzelle wird vom Betriebssystem benutzt, um festzustellen, ob während einer RS232-Datenübertragung Bits verloren gingen. Da derartige Prüfungen mit Parity-Bits (Quersummenprüfung) des öfteren erwähnt werden, gebe ich eine kurze Beschreibung des Prüfprinzips im Texteinschub 18 »Fehlererkennung mit Parity-Bits«.

Zusätzlich wird in 171 die Länge des Band-Vorspanns bei seiner Erzeugung gezählt.

Adresse 172 und 173 ($AC und $AD)

Zeiger auf die Anfangsadresse für Ein-/Ausgabe, Zwischenspeicher für den Bildschirmeditor

In den Speicherzellen 193 und 194 steht ein Zeiger, der auf die Adresse im Programmspeicher zeigt, wo das Programm beginnt beziehungsweise beginnen soll, welches abgespeichert beziehungsweise geladen werden soll.

Dieser Zeiger wird am Anfang einer Lade- oder Abspeicher-Operation in die Zellen 172 und 173 gebracht, wo er während der Operation laufend erhöht wird, bis das Ende des Programms erreicht ist; dann wird er wieder auf seinen ursprünglichen Wert gesetzt.

Der Zeiger dient außerdem noch dem Bildschirmeditor als Zwischenspeicher während des Scrollens (Hochschieben) des Bildschirms und beim Einfügen zusätzlicher Zeilen.

Dieser Zeiger kann sehr nützlich sein, um Programme entweder schon beim SAVEn oder aber erst beim LOADen gezielt auf andere als ursprünglich verwendete Speicherbereiche zu bringen. Dazu sind aber noch einige andere Zellen notwendig, bis hin zu dem schon erwähnten Zeiger in 193 und 194.

Adresse 174 und 175 ($AE und $AF)

Zeiger auf die Endadresse für Ein-/Ausgabe, Zwischenspeicher für den Bildschirmeditor

Dieser Zeiger ist der Zwilling zu 172 und 173, nur zeigt er seinerseits auf die letzte Adresse des zu bewegenden Programms.

Adresse 176 und 177 ($B0 und $B1)

Zeitkonstante

Der Wert in dieser Speicherzelle wird verwendet, um die Zeitkonstante zum Lesen vom Band in der Zelle 146 einzustellen.

Adresse 178 und 179 ($B2 und $B3)

Zeiger auf den Kassetten-Puffer

Beim Einschalten des Computers werden diese Speicherzellen in Low-/High-Byte-Darstellung auf die Anfangsadresse des Kassetten-Puffers gesetzt. Beim VC 20 und C 64 ist dies die Adresse 828 ($33C).

Durch Verbiegen dieses Zeigers kann der Kassettenpuffer auf beliebige Plätze des Speichers, aber nicht unterhalb der Adresse 512 verschoben werden. Das kann durchaus sinnvoll sein, um im Kassettenpuffer abgelegte Maschinenprogramme vor Überschreiben durch Kassettenoperationen zu schützen.

Adresse 180 ($B4)

RS232-Bit-Zähler und -Zwischenspeicher für Kassetten-Operationen

Die RS232-Routinen verwenden die Speicherzelle 180, um die Zahl der übertragenen Bits zu zählen, außerdem für Parity-Berechnung (siehe Texteinschub 18) und Stop-Bit-Bearbeitung.

Die Lade-Routinen für Kassettenbetrieb benutzen diese Zelle als Flagge, die angibt, ob der Computer bereit ist, Daten zu übernehmen.

Adresse 181 ($B5)

RS232-Anzeige für nächstes Bit, Flagge für End-of-Tape

Bei RS232-Operationen enthält die Zelle 181 das jeweils nächste Bit, welches übertragen werden soll. Bandoperationen entnehmen dieser Speicherzelle, welcher Block gerade gelesen wird.

Adresse 182 ($B6)

Ausgabe-Zwischenspeicher für RS232 und Kassette

Bei Ausgabe von Daten über die RS232-Schnittstelle wird jedes Byte in seine Einzelteile zerlegt, bevor es über den Ausgabepuffer seriell übertragen wird. Der Ausgabepuffer wird im obersten Teil des Programmspeichers angelegt (siehe auch Speicherzellen 55 und 56); die genaue Anfangsadresse steht in Speicherzelle 248. Auch die Ausgabe von Daten auf die Kassette verwendet Zelle 182 als Ausgabe-Zwischenspeicher.

Texteinschub Nr. 16
Die eingebaute Uhr

Der VC 20 und der C 64 haben eine interne Uhr eingebaut, deren Stand abgefragt, ausgedruckt und somit zur Zeitmessung und Programmsteuerung eingesetzt werden kann.

In der Basic-Befehlsliste der Handbücher finden wir dazu zwei Funktionen, TI und TI$.

1) TI gibt den Stand des Zählers wieder, der durch die drei Spei-

cherzellen 160, 161 und 162 gebildet wird. Dabei ist der Wert von TI nichts anderes als die Summierung des Inhalts dieser drei Zähler.

Entsprechend dem dreistelligen Zählerprinzip (siehe Beschreibung der Speicherzellen 160 bis 162) ist die Summe:

```
TI    = Inhalt (162)
      + Inhalt (161) * 256
      + Inhalt (160) * 256 * 256
```

Mit dem folgenden kleinen Programm können wir das verifizieren:

```
10    PRINT TI;
20    PRINT PEEK(162)+256*PEEK(161)+256*256*PEEK(160)
30    GOTO 10
```

Die beiden Zahlenbänder für TI und die Zählersumme sind praktisch identisch.

**2) TI$** gibt ebenfalls den Stand des Zählers wieder, aber in einer anderen Darstellung. Während TI 60mal in der Sekunde weiterzählt, gibt TI$ direkt Stunden, Minuten und Sekunden an.

Den Zusammenhang zwischen TI und TI$ können Sie am besten mit dem folgenden kleinen Programm sehen:

```
10    PRINT INT(TI/60);
20    PRINT TI$
30    GOTO 10
```

Zeile 10 rechnet TI in Sekunden um. Damit die Zeile nicht mit vielen Dezimalstellen volläuft, verwandelt sie das Resultat in eine ganze Zahl. Zeile 20 zeigt dazu im Vergleich die sechs Ziffern von TI$.

Das erste, was beim Ablauf des Programms auffällt, ist die gleichzeitige Umschaltung beider Zahlenreihen. Die Umrechnung von TI$ nach TI geht am besten »zu Fuß«. Stoppen Sie den Lauf mit der STOP-Taste. Nehmen Sie dann den letzten Wert von TI$ (rechts). Die ersten beiden Ziffern sind die Stunden, ihr Wert wird mit 3600 multipliziert, um sie in Sekunden umzurechnen. Addieren Sie dazu den Wert der mittleren beiden Ziffern (Minuten) multipliziert mit 60, und addieren Sie zu diesem Zwischenergebnis die Sekunden (Ziffern ganz rechts). Das Resultat ist identisch mit dem letzten Wert von TI.

Wenn Sie übrigens den Ausdruck für TI$ optisch verbessern wollen, dann setzen Sie zwischen die Stunden, Minuten und Sekunden einen Doppelpunkt. Das wird durch eine String-Manipulation erreicht:

```
Print LEFT$(TI$,2)":"MID$(TI$,3,2)":"RIGHT$(TI$,2)
```

Eine gute Uhr muß sich stellen lassen – bei TI$ erreichen wir das einfach mit Zuweisen des gewünschten Wertes an die Variable TI$. Zum Beispiel stellt

```
TI$ = "153000"
```

die Uhr auf 15 Uhr 30. Man kann sie dementsprechend auch auf 0 zurücksetzen, was bei einem Stoppuhr-Betrieb notwendig wird.

TI kann direkt nicht beeinflußt werden, nur über POKEn von neuen Werten in die Speicherzellen 160 bis 162 oder durch die Zuweisung von Werten an TI$.

Die eleganteste Methode, TI und TI$ auf 0 zu setzen, geht beim C 64 und VC 20 mit

```
SYS 65499
```

Wenn Sie noch das kleine Programm von oben im Rechner haben, können Sie es gleich ausprobieren. Geben Sie direkt ein:

```
SYS 65499:RUN
```

und die Uhr startet von Null an.

Abschließend möchte ich Ihnen noch zwei kleine Anwendungsbeispiele von TI und TI$ mitgeben. Das erste ist ein Kochrezept, wie die Laufzeit eines Programms gemessen werden kann. Diese Programm-Stoppuhr besteht aus zwei Zeilen.

Die erste Zeile setzt die Uhr auf 0, das kennen wir schon.

Die zweite Zeile druckt am Ende des Programms die abgelaufene Zeit aus.

```
10 TI$ = "000000"
:
:
10000 PRINT TI/60 "SEKUNDEN"
```

Das zu messende Programm steht zwischen diesen beiden Zeilen.

Das zweite Beispiel betrifft eine Uhr, die nach einer vorgegebenen Zeit ein Programm (Spiel) abbricht. Davon zeige ich zwei Versionen. Die eine Version ist nach allen Erklärungen von oben beinahe trivial:

```
10 TI$ = "000000"
1000 IF TI$ > "000700" THEN STOP
```

Diese beiden Zeilen setzen die Uhr auf 0 und brechen ein Programm nach genau 7 Minuten ab.

Etwas kniffliger ist der Abbruch (oder Start) mit einer Countdown-Uhr.

```
10    TI$ = "000000"
20    ZEIT = 300
30    IF ZEIT-VAL(TI$) <=0 THEN STOP
40    weiteres Programm
```

Die Variable »Zeit« gibt die Dauer des Countdown in Sekunden an. Zeile 30 überprüft den Wert von TI$, bis er 300 erreicht hat, indem sie den jeweiligen Wert von TI$ von der vorgegebenen Zeit subtrahiert. Natürlich müssen in beiden Versionen die Prüfzeilen sinnvoll in ein Programm eingebaut werden. Aber das möchte ich gern Ihnen überlassen.

Texteinschub Nr. 17
Experimente mit dem Kassetten-Puffer

Die Speicherzellen von 828 bis 1 019 werden als »Kassetten-Puffer« bezeichnet.

Beim Speichern auf eine Kassette wird zuerst der Vorspann eines Bandes, der sogenannte »Header«, in diesen Puffer gespeichert. Ein Programm wird dann direkt auf das Band geschrieben. Eine Datei allerdings läuft zuerst auch in den Kassetten-Puffer und von dort erst auf das Band. Sie kennen sicher die charakteristischen Wartezeiten des Kassettenmotors beim SAVEn einer Datei.

Beim Laden von einer Kassette gilt der Unterschied zwischen einem Programm und einer Datei genauso, einschließlich der Benutzung des Kassetten-Puffers.

Wir haben gelernt, daß in der Speicherzelle 166 die Zahl der Bytes gezählt wird, die in den Puffer geschrieben beziehungsweise aus dem Puffer gelesen worden sind. Die Zahl reicht von 0 bis 191.

Diese Speicherzelle 166 kann während eines Programms abgefragt und auch mit POKE beliebig verändert werden. Was dabei herauskommt, ist vordergründig nur eine Spielerei. Aber vielleicht kann man die folgenden Experimente auch nutzbringend einsetzen.

Zuerst wollen wir die Funktionsweise von 166 erproben. Dazu laden wir eine simple Datei auf ein leeres Band, und zwar mit folgendem Programm:

Programm # 1

```
10    OPEN 1,1,1
20    FOR I=100 TO 150
30    PRINT #1,1
40    NEXT
50    CLOSE 1
```

Wir eröffnen eine Datei (ohne Namen) mit der Nummer 1, für Kassette (die zweite 1), zum Schreiben (die dritte 1). Nach RUN wird der Kassetten-Puffer mit den Zahlen 100 bis 150 in mehreren Schüben gefüllt, wobei jeder Schub einzeln auf das Band geschrieben wird.

Den Zusammenhang zwischen den Datei-Zahlen und dem Zähler in 166 zeigt uns das folgende Ausleseprogramm:

Programm # 2

```
10    OPEN 1,1,0
20    GET #1,X$
30    Print X$;
40    PRINT CHR$(28)PEEK(166)CHR$(154);
50    GOTO 20
```

Wir eröffnen wieder eine Datei, diesmal zum Lesen (die 0), und bringen mit GET# die einzelnen Zeichen hintereinander in den

Puffer und dann auf den Bildschirm. Die Zeile 40 druckt nach jedem Zeichen in roter Farbe [CHR$(28)] den Zählerstand und schaltet dann mit CHR$(154) – beim VC 20 wäre das CHR$(31) – wieder auf die Normalfarbe zurück.

Zuerst muß das Band zurückgespult werden, und dann geht es los mit RUN. Nach dem Erscheinen der ersten Zeichen auf dem Bildschirm stoppen Sie bitte den Ablauf mit der STOP-Taste.

Sie sehen jetzt in Rot den Inhalt der Zelle 166, die aufwärts zählt, und dazwischen in Blau die Zahlen von 100 aufwärts. Interessant ist, daß durch Zwischenräume für eine 3stellige Zahl 6 Byte verbraucht werden.

Fahren Sie mit CONT so lange fort, bis der Kassettenmotor anläuft und der nächste Schub auf dem Bildschirm ausgedruckt wird. Nach erneutem STOP sehen Sie, daß die roten Zahlen nach 190 wieder auf 0 zurückspringen. Das war der Moment, wo der Kassettenmotor wieder eingeschaltet wurde.

Diese Erkenntnis verwenden wir für ein Experiment.

Mit der in das Programm # 2 eingeschobenen Zeile 45 fragen wir den Inhalt von 166 ab und beeinflussen damit den Ablauf des Programms. Außerdem setzen wir an dieser Abfragestelle den Inhalt der Zelle 166 auf den Endwert 191 und zwingen damit den Kassettenmotor weiterzulaufen.

```
45 IF PEEK(166) = 18 THEN POKE 166,191
```

Die Wiederholung des Programms mit zurückgespultem Band bringt uns ein neues Ergebnis:

Sobald der Zähler in 166 die 18 erreicht hat, glaubt das Programm, der Kassetten-Puffer wäre bereits ausgelesen, schaltet den Kassettenmotor wieder ein und liest den nächsten Zahlenblock in den Puffer. Wir erhalten jetzt nicht alle Zahlen, die auf dem Band stehen, sondern nur Gruppen von 18 Byte, das sind ungefähr drei Zahlen.

Ich sage »ungefähr« mit Absicht, denn mit der Symmetrie beziehungsweise mit der richtigen Reihenfolge klappt es nicht immer so ganz, da ja die Länge des Kassetten-Puffers nicht unbedingt ein ganzzahliges Vielfaches der ausgelesenen Bytes ist. Da liegt also ein kleines Problem.

Dieses Abfragen und Abändern der Speicherzelle 166 geht natürlich auch in der anderen Richtung, nämlich beim Abspeichern von Zahlen. Nehmen Sie bitte noch mal das Programm # 1 her und ändern Sie es wie folgt ab:

Programm # 1.a

```
10  OPEN 1,1,1
20  FOR I=100 TO 300
30  PRINT#1,1
35  IF PEEK(166)=18 THEN POKE 166,191
40  NEXT
50  CLOSE 1
```

Wir haben jetzt die Abfrage der Speicherzelle 166 des Programms # 2 von vorhin in das Programm # 1 eingebaut. Spulen Sie bitte das Band zurück, und lassen Sie das Programm laufen.

Nun wollen wir die dadurch neu abgespeicherte Datei ganz normal auslesen. Dazu nehmen wir das Programm #2, also ohne die Zeile 45. Das sieht dann so aus:

Programm #2.a

```
10  OPEN 1,1,0
20  GET#1,X$
30  PRINT X$;
40  PRINT CHR$(28)PEEK(166)CHR$(154);
50  GOTO 20
```

Wir starten es mit RUN, nachdem das Band wieder zurückgespult ist. Der Vorgang ist im Prinzip der gleiche wie bei Programm #2; halten Sie das Programm bitte auch wieder an, so wie vorher.

Wir sehen aber einen großen Unterschied im Ausdruck. Es erscheinen nur die ersten drei Zahlen, 100 bis 102, danach steht nichts mehr im ganzen Block, bis der Inhalt von 166 die Endzahl 190 erreicht hat. Erst danach, nach dem Loslaufen des Kassettenmotors und dem Einlesen des nächsten Schubes, erscheinen die nächsten drei Zahlen.

Schlußfolgerung:

Durch POKEn der Zahl 191 in die Speicherzelle 166 zu einem beliebigen Zeitpunkt können wir sowohl beim Speichern als auch beim Einlesen einer Datei dem Computer vorgaukeln, der Kassetten-Puffer sei bereits abgearbeitet. Dadurch wird der Kassettenmotor eingeschaltet und der nächste Schub ein- beziehungsweise ausgelesen.

Texteinschub Nr. 18
Fehlererkennung mit Parity-Bits

Bei der Datenübertragung zwischen Peripheriegeräten, insbesondere zwischen Datasette und dem Computer, kommt es recht häufig vor, daß Fehler auftreten. Diese Fehler haben alle möglichen Ursachen, und trotz aller Anstrengungen der Ingenieure lassen sie sich leider nicht völlig vermeiden.

An besonderen Schwachstellen werden daher Maßnahmen getroffen, um Fehler wenigstens zu erkennen und Programme abzubrechen, bevor größerer Schaden entsteht. Die mißlichen »LOAD ERROR«-Meldungen sprechen da eine deutliche Sprache.

Die einfachste Art, Fehler zu erkennen – ich sollte genauer sagen: einzelne Bitfehler zu erkennen –, geschieht über sogenannte »Parity-Bits«. Die Methode besteht darin, daß zu einem Datenwort, zum Beispiel einem Byte, ein zusätzliches Bit hinzugefügt wird, und zwar so, daß die Quersumme immer eine gerade oder auch eine ungerade Zahl ergibt.

Im binären Zahlensystem sieht das so aus:

DEZ	BINÄR	PARITYBIT	QUERSUMME
0	0000	0	0
1	0001	1	0
2	0010	1	0
3	0011	0	0
4	0100	1	0
.	.	.	.
.	.	.	.

Bevor ein Wort übertragen wird, errechnet der Sender das Parity-Bit und fügt es dem Wort als zusätzliches Bit hinzu. Der Empfänger, der diese Prüfmethode natürlich auch kennen muß, rechnet die Quersumme aus. Wenn sie stimmt, nimmt er das Parity-Bit weg und arbeitet mit dem richtigen Wort weiter. Wenn die Quersumme nicht stimmt, schlägt er Alarm.

Sie werden sicher schon bemerkt haben, daß in meinem Beispiel natürlich ein Doppelfehler, nämlich zwei falsche Bits, natürlich nicht erkannt werden. Um das zu erreichen, müßte man zwei Parity-Bits einführen.

Sie sehen natürlich auch, wohin das letztlich führt, nämlich zu einer Vergrößerung der Wortlänge. Man nennt das auch »Redundanz«, vielleicht haben Sie dieses Wort schon einmal gehört. Nun, da gibt es für jeden Anwendungsfall ein Optimum, abhängig von der Wahrscheinlichkeit, welche Art von Fehlern in welcher Häufigkeit auftreten. Im Extremfall gibt es Codiersysteme – zu denen die Parity-Bit-Methode auch gehört –, welche in der Lage sind, Fehler nicht nur zu erkennen, sondern gleich zu korrigieren.

Texteinschub Nr. 19
Files – Geräte – Namen – Nummern

In den Handbüchern von Commodore und auch in anderen Beschreibungen wird von den Ein- und Ausgabe-Befehlen, wie zum Beispiel LOAD, SAVE, OPEN etc., leider ein recht verwirrendes Bild geboten. Ich beziehe mich dabei auf die hinter diesen Befehlen stehenden Ziffern und Namen.

In der Bezeichnung »File-Name« und »Geräte-Nummer« sind sich die Autoren noch weitgehend einig. Daß die dritte Angabe hinter den Befehlen aber Sekundär-Adresse, Command, Speicheradressen-Flag oder gar EOT-Flag genannt wird, muß den Computeramateur zwangsläufig verwirren.

Da wir im nebenstehenden Kurs gerade die Speicherzellen 183 bis 188 besprechen, die alle mit diesen Anhängseln der eingangs genannten Befehle zusammenhängen, ist dies eine gute Gelegenheit, etwas Systematik in die Angelegenheit zu bringen.

Als erstes gebe ich die Bezeichnungen der Befehle aus einem Commodore-Buch wieder, bevor die einzelnen Angaben im Detail diskutiert werden.

- LOAD "File-Name", Geräte-Nr., Speicheradressen-Flag
- SAVE "File-Name", Geräte-Nr., EOT-Flag
- VERIFY "File-Name", Geräte-Nr., Speicheradressen-Flag
- OPEN File-Nr., Geräte-Nr., Sekundär-Adresse, "File-Name, Typ, Modus"
- INPUT# File-Nr., Variable
- GET# File-Nr., Variable
- CMD File-Nr.
- CLOSE File-Nr.

File-Name

»File« wird normalerweise mit »Datei« übersetzt.

Einen Datei-Namen gibt es aber nur beim OPEN-Befehl. Bei den Befehlen LOAD, SAVE und VERIFY ist der Name des Programms gemeint, der bekanntlich in Gänsefüßchen hinter diesen Befehlen steht. Bei Disketten als Pflicht, bei Kassetten als Option. Beim OPEN-Befehl steht der Name zwar auch in Gänsefüßchen, aber nicht direkt hinter dem Befehl, sondern erst an vierter Stelle. Auch er kann bei Kassetten-Betrieb, aber auch beim Drucker weggelassen werden.

Bei Disketten-Dateien stehen hinter dem Namen – als sein Bestandteil – noch Typ und Modus. Typ bezeichnet die Art der Datei (RELative, SEQuential, USeR, PRoGramm), Modus die Operation (Read, Write, Append).

Der File-Name kann bei Disketten auch eine Anweisung an das Betriebssystem des Disketten-Laufwerks sein (NEW, SCRATCH, RENAME etc.). Nähere Erklärungen dazu finden Sie im Handbuch des Disketten-Laufwerks.

Da in 187/188 die Adresse angegeben ist, ab der der Programm- beziehungsweise Datei-Name im Speicher steht, können wir ihn dort ansehen.

Geben Sie direkt ein

```
LOAD "1234"
```

Nach der RETURN-Taste kommt die übliche Aufforderung der Datasette, die wir aber mit der STOP-Taste abwürgen. In 187/188 steht jetzt die Anfangsadresse »A« des gespeicherten Namens:

```
A = PEEK(187)+256*PEEK(188):PRINT PEEK(A)
```

Wir erhalten die Zahl 49, das ist der ASCII-Codewert der Ziffer »1«. Durch Erhöhen der Anfangsadresse »A« im PRINT-Befehl um jeweils 1 kommen auch die übrigen Zeichen des Namens zum Vorschein.

Die VC 20-Besitzer kennen sicher den Effekt, der entsteht, wenn beim SAVE-Befehl vor dem File-Namen, aber innerhalb der Gänsefüßchen, eine Farbtaste eingegeben wird. Die FOUND-Meldung druckt nämlich den File-Namen in der gewählten Farbe aus, was besonders bei Kassetten-Operationen sehr eindrucksvoll ist.

Das Disketten-Laufwerk bietet eine andere Überraschung. Die geSHIFTete SPACE-Taste innerhalb des File-Namens erzeugt das zweite Gänsefüßchen.

```
SAVE"SPIEL 2 (SHIFT-SPACE) SYS 16000",8
```

erscheint in der Directory als:

```
"SPIEL 2" SYS 16000
```

und kann mit LOAD"SPIEL 2",8 geladen werden.

Auf diese Weise können Sie in einer gut lesbaren Art Anweisungen in den Namen eines Programms einbauen. Vorsicht ist allerdings geboten, wenn Sie die SHIFT-SPACE-Taste vor dem File-Namen drücken. Dann wird nämlich der 2. Gänsefuß direkt nach dem ersten gesetzt, und der nachfolgende File-Name steht außerhalb der Gänsefüße. Ein derartiges Programm ist von Uneingeweihten nur sehr schwer zu laden.

Geräte-Nummer

Alle an den Computer anschließbaren Peripheriegeräte haben eine eigene Nummer, unter welcher sie »angesprochen« werden können. Es handelt sich eigentlich um eine Adresse, und in der Tat sprechen manche Autoren von der »Primär-Adresse«. In der Beschreibung der Speicherzelle 184 sind sie alle zusammengestellt.

Bei den Befehlen LOAD, SAVE, VERIFY und OPEN steht diese Geräte-Nummer als zweite Angabe, vor der ersten Angabe durch ein Komma getrennt:

LOAD"NAME",1.....bedeutet "Laden von Datasette"
LOAD"NAME",8.....bedeutet "Laden von Diskette"
OPEN 12,4.....bedeutet "An den Drucker"

(Die 12 hinter dem OPEN-Befehl ist eine »File-Nummer«. Sie kommt weiter unten an die Reihe.)

Die Geräte-Nummer kann auch weggelassen werden. Dann allerdings nimmt der Computer automatisch an, daß es sich um die Datasette handelt. Die Geräte-Nummer steht jeweils in der Speicherzelle 186 und kann von dort mit PEEK ausgelesen werden.

Sekundär-Adresse

Die dritte Angabe hinter LOAD, SAVE und OPEN hat widersprüchliche Namen, wohl deshalb, weil sie abhängig von der Gerätenummer verschiedene Aufgaben hat. Von den ganz am Anfang schon genannten Varianten gefällt mir »Sekundär-Adresse« am besten.

Tabelle 6 faßt die Funktionen der Sekundär-Adresse zusammen.

Befehl	Sekundäradresse	Funktion
LOAD	0	lädt Programm an den Anfang des Programmspeichers
	1	lädt ein Programm absolut, also an die Adresse, von der ab es gespeichert wurde.
SAVE	0	Normales SAVE, Programm wird bei späterem LOAD an den Anfang des Programm-Speichers geladen (Basic-Programme)
	1	erzwingt bei späterem LOAD des Programms die Speicherung ab der Adresse, wo es zur Zeit steht (Maschinen-Programme)
	2	setzt am Programmende auf der Kassette eine »Band-Ende«-Markierung, die beim »Überlesen« das Band mit der Fehlermeldung »DEVICE NOT PRESENT« stoppt.
	3	Kombination von 1 und 2
OPEN bei Kassette	0	Daten lesen
	1	Daten schreiben
	2	Daten schreiben mit »Bandende«-Markierung
OPEN bei Diskette	0	vom Betriebssystem der Floppy für »Laden« reserviert
bei Diskette	1	vom Betriebssystem der Floppy für »Speichern« reserviert
	2–14	reserviert numerierten Daten-Kanal, bis zu drei gleichzeitig
	15	reserviert Kommando-Kanal (nähere Angaben zu diesen Sekundär-Adressen siehe Floppy-Handbuch)
OPEN bei Drucker	0–10	die Funktionen sind bei den Druckern zum Teil verschieden. Bitte in Drucker-Anleitung nachsehen.

Tabelle 6. Funktionen der Sekundäradressen

Sie sehen, die Sekundär-Adressen haben es in sich! Die gerade benutzte Sekundär-Adresse steht in der Speicherzelle 185 und kann mit PEEK von dort ausgelesen werden.

File-Nummer

Die File-Nummer, oft auch logische File-Nummer genannt, steht als erste Angabe hinter dem OPEN-Befehl und den damit verbundenen PRINT#-, INPUT#-, GET#- und CLOSE-Befehlen.

Sie gibt einer zu bearbeitenden Datei eine Nummer, die von den nachfolgenden, anderen Befehlen ebenfalls verwendet werden muß, wenn sie sich auf dieselbe Datei beziehen. Auf diese Weise können mehrere Dateien nebeneinander bearbeitet werden, und zwar fünf bei Disketten und auf alle Geräte verteilt insgesamt zehn.

File-Nummern dürfen die Werte von 1 bis 255 haben. Bei Werten über 127 wird bei einem PRINT#-Befehl nach jedem RETURN-Zeichen – CHR$(13) – zusätzlich ein ASCII-Code für Zeilenvorschub – CHR$(10) – an das Gerät gegeben. Diese Eigenschaft kann bei denjenigen Geräten nützlich sein, die normalerweise auf CHR$(13) ohne Zeilenvorschub reagieren (kein automatisches Line-Feed).

Die File-Nummer steht jeweils in der Speicherstelle 184, von wo sie mit PEEK ausgelesen werden kann. Zum Beispiel, um nachzuprüfen, welches Gerät als letztes angesprochen worden ist.

Abschließend möchte ich nochmals darauf hinweisen, daß in den Speicherzellen 183 bis 188 immer die gerade als letzte verwendete Angabe steht. Wir haben aber gesehen, daß der Computer sich maximal zehn File-Nummern mit dazugehörenden Geräte-Nummern, Sekundär-Adressen etc. merken kann.

Er tut dies in speziellen Tabellen, die in den Speicherzellen 601 bis 630 stehen.

Wir werden uns also noch einmal mit dieser Sache befassen, wenn wir bei den entsprechenden Adressen angekommen sind.

Adresse 183 ($B7)

Länge des derzeitigen File-Namens

Die LOAD-, SAVE- und VERIFY-Befehle für Disketten verlangen die Angabe eines Programm- oder Dateinamens, auf Computerdeutsch »File-Name«. Nähere Angaben dazu finden Sie im Texteinschub Nr. 19 »Files – Geräte – Namen – Nummern«.

Auch der OPEN-Befehl kann einen File-Namen haben. Bei Kassettenoperationen kann der File-Name weggelassen werden.

In der Speicherzelle 183 steht während und nach der Verwendung eines der oben genannten Befehle eine Zahl, die angibt, aus wie vielen Zeichen der File-Name besteht.

Bei Disketten sind File-Namen möglich, die aus maximal 16 Zeichen bestehen.

Bei Kassetten dagegen sind Namenslängen von maximal 187 Zeichen erlaubt. Allerdings werden vom Computer auf dem Bildschirm nur 16 Zeichen ausgedruckt (siehe dazu den Texteinschub 20 »Tape-Header«).

Für die Längenangabe in Zelle 183 gilt dabei nur die Anzahl derjenigen Zeichen, die zwischen den Gänsefüßchen stehen.

Diese Zahl kann nach einer Ein-/Ausgabeoperation, auch nach einer ungültigen oder abgebrochenen, durch PEEK (183) ausgelesen werden.

Ein File-Name wird übrigens auch bei einem OPEN-Befehl der RS232-Schnittstelle angegeben. Dieser Name, der bis zu vier Zeichen lang sein kann, wird in die Speicherzellen 659 bis 662 übertragen und gibt dort die Übertragungsrate, Wortlänge und Parity-Prüfung an.

Adresse 184 ($B8)

Nummer der derzeitigen Datei (File)

Hinter jedem OPEN-Befehl steht eine Zahl, die der durch diesen Befehl angefangenen Datei zugeordnet wird. Diese Datei- oder File-Nummer gilt als Referenz für alle anderen Ein- und Ausgabebefehle derselben Datei. Nähere Angaben dazu können Sie dem Texteinschub Nr. 19 »Files – Geräte – Namen – Nummern« entnehmen.

Ein OPEN-Befehl ruft die entsprechende Routine des Betriebssystems auf, welche die File-Nummer in die Speicherzelle 184 schreibt. Vor dort kann sie mit PEEK(184) ausgelesen werden. Geben Sie die folgende Zeile direkt ein:

```
A=30:OPEN A,3:PRINT PEEK
(184):CLOSE A
```

Um verschiedene File-Nummern auszuprobieren, definieren wir sie als Variable A. Nach dem »A« des OPEN-Befehls steht die Zahl 3. Damit wird der Bildschirm angewählt (siehe »Sekundär-Adresse« im schon erwähnten Texteinschub). Das Anwählen des Bildschirms vermeidet eine störende Meldung des Betriebssystems.

Mit RETURN nach der obenstehenden Zeile wird der jeweilige Wert von A als Inhalt der Zelle 184 ausgedruckt.

Adresse 185 ($B9)

Derzeitige Sekundär-Adresse

Die Sekundär-Adresse steht als dritte Angabe hinter den Ein- und Ausgabe-Befehlen LOAD, SAVE, VERIFY und OPEN. Sie hat bei den verschiedenen Peripheriegeräten spezielle Funktionen. Diese Funktionen sind im Texteinschub 19 näher erläutert.

Der jeweilige Wert der Sekundär-Adresse steht in der Speicherzelle 185, allerdings um 96 erhöht. Für Sekundär-Adressen stehen, über die Standardwerte der einzelnen Peripheriegeräte hinaus, die Zahlen von 0 bis 31 zur Verfügung. Ab 32 fängt in Zelle 185 wieder der Zyklus ab 0 an. Das können wir uns anschauen. Ich wähle zur Eröffnung einer Datei wieder den Bildschirm als »nicht-störendes« Empfangsgerät.

```
A=15:OPEN 1,3,A:PRINT
PEEK(185)-96:CLOSE 1
```

Durch Verändern des Wertes von A können Sie alle Möglichkeiten durchspielen.

Adresse 186 ($BA)

Derzeitige Geräte-Nummer

Jedes an den Computer anschließbare Gerät hat eine eigene Nummer, die zusammen mit den Ein-/Ausgabe-Befehlen LOAD, SAVE, VERIFY und OPEN angegeben werden muß. Wird keine Nummer angegeben, nimmt der Computer automatisch an, daß die Datasette gemeint ist.

Alle von Commodore vorgegebenen Geräte-Nummern sind in der folgenden Tabelle 5 aufgelistet.

Geräte-Nummer	angesprochenes Gerät
0	Tastatur
1	Datasette
2	RS232- (User-Port) Schnittstelle
3	Bildschirm
4	Drucker (normal)
5	Drucker (zusätzlich)
8	Disketten-Laufwerk Nr. 0
9	Disketten-Laufwerk Nr. 1
10, 11	weitere Disketten-Laufwerke

Tabelle 5. Von Commodore vorgegebene Geräte-Nummern

Die normale Geräte-Nummer eines Druckers ist 4, die eines Disketten-Laufwerks 8. Die zusätzlichen Nummern müssen gesondert am betreffenden Gerät eingestellt werden.

Nach der Ausführung eines der oben genannten Befehle steht die entsprechende Geräte-Nummer in der Speicherzelle 186, aus der sie mit PEEK(186) ausgelesen werden kann.

Adresse 187 und 188 ($BB und $BC)

Zeiger auf Adresse des derzeitigen File-Namens

Die Bedeutung eines Programm- oder Dateinamens – normalerweise kurz »File-Name« genannt – ist im Texteinschub Nr. 19 »File – Geräte – Namen – Nummern« näher beschrieben. In den Speicherzellen 187 und 188 steht in der Low-/High-Byte-Darstellung ein Zeiger auf diejenige Adresse im Programm-Speicher, wo dieser Name gespeichert ist.

Eine Ausnahme ist hier der OPEN-Befehl der RS232-Schnittstelle. Ihr File-Name wird in die Speicherzellen 659 bis 662 gebracht, wo er verschiedene Parameter dieser Schnittstelle steuert.

Adresse 189 ($BD)

Zwischenspeicher für RS232-Parity-Prüfung und für Kassettenoperationen

Die RS232-Routinen benutzen diese Speicherzellen als Zwischenspeicher für ein Prüf-Byte (Parity-Prüfung) bei der Ausgabe. Die Parity-Prüfung habe ich kurz im Texteinschub Nr. 18 erklärt.

Auch die Kassetten-Routinen bedienen sich dieser Speicherzelle. Sie verwenden sie als Zwischenspeicher für das gerade

gesendete oder empfangene Zeichen.

Adresse 190 ($BE)

Blockzähler für Kassetten-Ein-/Ausgabe

Das Betriebssystem des Computers schreibt bei SAVE ein Programm zweimal auf das Band der Datasette. Beim LOAD-Befehl wird der erste Block in den Arbeitsspeicher des Computers geladen; der zweite – identische – Block wird dann mit dem ersten Block Byte für Byte verglichen, um Datenfehler auf dem nicht immer ganz zuverlässigen Bandmaterial zu erkennen.

In der Speicherzelle 190 wird dem Betriebssystem angezeigt, wie viele Blockteile bei diesem Prozeß noch gelesen oder gespeichert werden müssen. Vom Basic-Programm aus ist diese Speicherzelle nicht zugänglich.

Adresse 191 ($BF)

Zwischenspeicher für LOAD-Operationen vom Band

Diese Speicherzelle wird beim Laden eines Programms vom Band dazu benutzt, um Zeichen aus einzelnen Bits zusammenzusetzen.

Adresse 192 ($C0)

Motorsperre der Datasette

Die Tasten der Datasette werden 60mal in der Sekunde von der »Interrupt-Routine« des Betriebssystem überprüft, ob eine von ihnen gedrückt worden ist. Die Speicherzelle 192 spielt dabei eine entscheidende Rolle, beim C 64 allerdings in einer anderen Weise als beim VC 20. Wie sie diese Rolle beim C 64 spielt, ist im Zusammenhang mit der Speicherstelle 1 ganz am Anfang dieses Kurses beschrieben worden. Ich habe dabei in zwei Beispielen gezeigt, wie durch Abfrage des vierten Bits von Adresse 1 geprüft werden kann, ob eine Taste der Datasette gedrückt ist und wie der Motor durch Setzen und Löschen des Bit 5 der Zelle 1 ein- und ausgeschaltet werden kann. Vorausgesetzt, der Inhalt der Speicherzelle 192 ist ungleich Null und eine Taste der Datasette ist gedrückt.

Nun will ich, wie versprochen, denselben Vorgang für den VC 20 beschreiben.

Wie Sie sich vielleicht noch erinnern, wird die Speicherzelle 1 beim VC 20 nicht für die Steuerung der Ein- und Ausgänge des Mikroprozessors verwendet. Diese Rolle wird beim VC 20 durch zwei Register des »Versatile Interface Adapter« (VIA 6522-A) ausgefüllt.

Für die Abfrage der Datasetten-Tasten ist das sechste Bit des VIA-Registers 37151 zuständig. Bei gedrückter Taste steht es auf 1, sonst auf 0. Ein kleines Programm zeigt es Ihnen:

```
10  X = PEEK(37151)
20  PRINT X
30  IF X = 62 THEN 50
40  GOTO 10
50  PRINT"TASTE
    GEDRÜCKT"
```

Wenn keine Taste gedrückt ist, läuft ein Zahlenband mit 126 ab. Die entsprechende Darstellung als Dualzahl lautet 1111 1110. Bei einer gedrückten Taste steht in 37151 die Zahl 62, als Dualzahl 0011 1110. Wichtig, wie gesagt ist nur das zweithöchste Bit.

Mit der Abfrage der Zeile 30 springt beim Drücken einer Taste das Programm auf die Zeile 50 und druckt den Text aus.

Den Motor der Datasette können wir mit Hilfe des Registers 37184 schalten. Wie beim C 64 gilt auch jetzt, daß dazu die hier angesprochene Speicherzelle 192, auch Interlock-Register genannt, eine Zahl größer als 0 enthält und daß außerdem eine Taste der Datasette gedrückt ist. Drücken Sie auf PLAY und geben Sie direkt ein:

```
POKE 192,255
POKE 37148,251: DER MOTOR
BLEIBT STEHEN.
POKE 37148,252: DER MOTOR
LÄUFT LOS.
```

Bestimmend sind hier Bit 2, 3 und 4.

Zum Ausschalten muß lediglich Bit 2 auf 1 stehen, zum Einschalten die drei Bits auf 110. Jede Zahl, die als Dualzahl diese Bedingungen erfüllt, kann dafür hergenommen werden. Um unabhängig von den anderen Bits des Registers 37148 zu bleiben, die ja auch ganz bestimmte andere Funktionen haben, empfiehlt es sich, über Boole'sche Verknüpfungen nur die wichtigen drei Bits zu verändern. Die beiden POKE-Befehle sehen dann so aus:

Ausschalten:

```
POKE 37148,PEEK(37148) OR 2
```

Einschalten:

```
POKE 37148,PEEK(37148)
AND 12
```

Adresse 193 und 194 ($C1 und $C2)

Anfangsadresse für Ein-/Ausgabe-Operationen

In diesen Speicherzellen steht in Low-/High-Byte-Darstellung die Adresse, ab der ein Programm gerade geladen oder gespeichert wird. Diese Adresse wird übrigens von hier auch in die Speicherzellen 172 und 173 gebracht, die wir schon früher besprochen haben.

Bei LOAD und SAVE auf Band steht hier die Anfangsadresse des Bandpuffers (828). Im Bandpuffer steht allerdings nur der sogenannte Bandvorspann (auf englisch »Tape Header«), während der Hauptteil des Programms im Programmspeicher ab einer Adresse steht, auf die der Zeiger in den Speicherzellen 195 und 196 hinweist.

Adresse 195 und 196 ($C3 und $C4)

Zeiger auf den Anfang des Programms hinter dem Tape Header

Bei jedem LOAD- und SAVE-Befehl für Kassetten wird der Vorspann (Tape Header), in dem Programmtyp, Anfangs- und Endadresse aufgezeichnet sind, im Kassettenpuffer ab Adresse 828 gespeichert. Der eigentliche Teil des Programms steht dann im Programmspeicher.

In den Speicherzellen 195 und 196 steht in der Low-/High-Byte-Darstellung diese Adresse, ab der das Programm beginnt. Ich habe für alle diejenigen, die mit der Datasette arbeiten, im Texteinschub Nr. 20 »Tape-Header« die Zusammenhänge mit einem Beispiel dargestellt.

Adresse 197 ($C5)

Tasten-Code der zuletzt gedrückten Taste

Bei der Behandlung der Speicherzelle 145 habe ich Ihnen mit Wort und Bild beschrieben, wie die Tasten des Computers abgefragt werden. Die dabei für jede Taste entstehende Dualzahl wird in eine Dezimalzahl (0 bis 63) umgewandelt und zuerst in die Speicherzellen 203 beziehungsweise 653 gebracht. Zur Umwandlung und Abfrage der Zellen 203 und 653 bringe ich bei diesen Speicherzellen mehr Details. Nach der Prüfung, welche Taste gedrückt worden ist, wird die Codezahl von 203 in die Speicherzelle 197 gebracht und dort »aufgehoben«. Diese vermeintliche Verdoppelung wird vom Betriebssystem dafür gebraucht, um zu erkennen, ob die nächste gedrückte Taste mit der vorhergehenden identisch ist. Ist sie identisch, dann entscheidet der Inhalt der Speicherzelle 650, ob das Zeichen dieser Taste mehrfach ausgedruckt wird. In 650 steht die sogenannte Wiederholfunktion. Aber ich will nicht vorgreifen. Die Codezahlen der einzelnen Tasten werde ich bei der Besprechung der Zelle 203 auflisten.

Adresse 198 ($C6)

Anzahl der Zeichen im Tastaturpuffer

Die Funktion des Tastaturpuffers, zu dem wir bei den Speicherzellen 631 und 640 noch kommen werden, habe ich bereits in diesem Kurs, und zwar im Texteinschub Nr. 15 »Dynamische Tastenabfrage« erklärt. Dabei habe ich damals schon sozusagen im Vorgriff die Zelle 198 verwendet.

In dieser Speicherzelle steht die jeweilige Anzahl der Zeichen, die im Tastaturpuffer gespeichert sind und darauf warten, weiterverarbeitet zu werden.

Das folgende kleine Programm zeigt es.

```
10  GET A$
20  PRINT PEEK (198);A$
30  FOR J=1 3000:NEXT J
40  GOTO 10
```

Der GET-Befehl holt ein Zeichen aus dem Tastaturpuffer – sofern eines dort zu finden ist. Die Zeile 20 druckt die Anzahl der Zeichen im Puffer aus, daneben das erste dieser Zeichen. Dann folgt eine Warteschleife, die uns erlaubt, ganz schnell ein paar Tasten zu drücken. Danach springt das Programm an den Anfang zurück und arbeitet diese eingegebenen Zeichen ab. Es ist dabei deutlich zu sehen, wie durch den GET-Befehl bereits ein Zeichen aus dem Puffer genommen und dadurch der Inhalt der Zelle 198 sofort um 1 reduziert wird.

Der Inhalt der Speicherzelle 198 kann mit POKE auch verändert werden.

Eine sinnvolle Anwendung dieser Beeinflussung erlaubt der nicht gerade sehr populäre WAIT-Befehl.

Ersetzen Sie bitte im obigen Programm die Warteschleife der Zeile 30 durch:

```
30 POKE 198,0: WAIT 198,1
```

Zuerst wird dem Computer vorgegaukelt, daß der Tastaturpuffer leer sei. Durch den WAIT-Befehl wartet das Programm danach so lange, bis ein Zeichen im Tastaturpuffer erscheint und springt erst dann auf die nächste Zeile 40.

Wenn Sie nach dem WAIT-Befehl statt der 1 eine 2 eingeben, wartet diese Zeile entsprechend auf zwei Tasteneingaben. Allerdings wird in der Zeile 20 dann nur jedes zweite Zeichen ausgedruckt.

Adresse 199 ($C7)

Flagge für reverse Darstellung der Zeichen

Normalerweise steht in dieser Speicherzelle eine 0, was mit PRINT PEEK (199) leicht nachgeprüft werden kann.

Sobald in der Zelle 199 eine andere Zahl als 0 steht, werden alle Zeichen in der reversen Darstellung gedruckt. Das Betriebssystem des Computers erhöht nämlich in diesem Fall den jeweiligen Bildschirmcode der Zeichen um 128. Ein Blick in eine Tabelle der Bildschirmcodes bestätigt, daß die Codes aller reversen Zeichen um genau 128 höher sind als die der normalen Zeichen.

Den reversen Modus können wir bekanntlich direkt mit der Kombination der CTRL- und der RVS-ON-Taste oder aber mit PRINT CHR$(18) herstellen. Wenn Sie aber versuchen sollten, das direkt einzugeben, um dann wieder mit PRINT PEEK

Texteinschub Nr. 20
Tape Header

Wenn ein Programm oder eine Datei auf Band gespeichert wird, setzt der Computer vor das Programm einen Vorspann, der auf englisch »Tape Header« genannt wird. Da dieser Name weit verbreitet ist, will ich ihn hier beibehalten. Der Tape Header 192 Byte lang. Er enthält alle wichtigen Angaben über das nachfolgende Programm.

Beim Laden eines Programms wird der Tape Header im Kassettenpuffer gespeichert, für den die Speicherstellen 828 bis 1019 reserviert sind. Von dort kann der Inhalt des Tape Headers gelesen und analysiert werden.

Bevor wir das versuchen, will ich erst seine Zusammensetzung erklären.

Im **ersten Byte** steht eine Kennzahl für den Typ des Programms. Diese Kennzahl ist abhängig von der Sekundär-Adresse, die beim SAVEn eingegeben worden ist. Die Arten der Sekundär-Adressen und ihre Bedeutung sind im Texteinschub Nr. 19 »Files – Geräte – Namen – Nummern« genau beschrieben. Es gibt Kennzahlen von 1 bis 5.

In Anlehnung an die Erklärung der Sekundär-Adresse kann man die Kennzahl generell dadurch beschreiben, daß ein Programm mit Kennzahl 1 immer an den Anfang des zur Verfügung stehenden Programm-Speichers geladen wird. Hauptsächlich kommt das für Basic-Programme in Frage.

Eine Kennzahl 3 bewirkt, daß das Programm an diejenige Stelle des Programmspeichers geladen wird, wo es vor dem SAVEn gestanden hat. Das ist hauptsächlich der Fall bei Programmen in Maschinensprache.

In Verbindung mit der Bedeutung der Sekundär-Adresse kann man den Zusammenhang, wie Tabelle 7 zeigt, darstellen.

Sekundär-Adresse	Kennzahl	Bedeutung
0, leer gerade	1	Programm wird ab Anfang des Basic-Speichers geladen
	2	Basic-Programm Datenblock, gefolgt von 191 Datenbytes
ungerade	3	Programm wird ab Adresse geladen, die in 829/830 steht
	4	Kopf für Basic-Programm Datenblöcke, die mit GET# etc. eingelesen werden
	5	logisches Bandende, das Betriebssystem sucht keine weiteren Programme

Tabelle 7. Tape Header Kennziffern

In **Byte 2 und 3** steht in Low-/High-Darstellung die Adresse, ab der das Programm im Speicher des Computers stand, als es gespeichert wurde.

In **Byte 4 und 5** steht die entsprechende End-Adresse des Programms.

Ab **Byte 6 bis Byte 192** steht der Name des Programms. Er darf also maximal 187 Zeichen lang sein. Bei LOAD werden allerdings nur 16 Zeichen auf dem Bildschirm dargestellt.

Jetzt wollen wir das alles mit einem kleinen Experiment überprüfen.

Schreiben Sie bitte ein kleines Programm, es braucht nicht sehr sinnvoll zu sein, wie zum Beispiel:

```
10   REM TAPE HEADER
20   REM TEST PROGRAMM
```

Nehmen Sie ein leeres Band und laden das Programm mit einem Namen, der länger sein soll als 16 Zeichen, zum Beispiel:

```
SAVE "TEST PROGRAMM FUER INHALT TAPE HEADER"
```

Nach Drücken der RECORD- und PLAY-Tasten der Datasette meldet der Computer:

```
FOUND TEST PROGRAMM FU
```

Es werden also nur 16 Zeichen inklusive Leerzeichen gedruckt. Sobald das Programm geladen ist, schauen wir im Kassettenpuffer nach, was in den ersten fünf Byte steht, danach lesen wir die restlichen Byte des Puffers.

Geben Sie direkt, ohne Zeilennummer, ein:

```
FOR I=0 TO 4: PRINT PEEK(828+I);: NEXT
```

Sie erhalten die Zahlen 1 1 8 41 8 (beim VC 20 mit 3-K-Speichererweiterung 1 1 4 41 4)

Danach geben wir wiederum direkt ein:

```
FOR I=5 TO 192: PRINT CHR$(PEEK(828+I));: NEXT
```

Beim VC 20 geben Sie in der FOR...NEXT-Schleife eine kleine Zahl ein, da der Bildschirmspeicher beim VC 20 kleiner ist als beim C 64.

Jetzt erscheint der volle Programmname, gefolgt von nicht sichtbaren Leerstellen. Wenn Sie in der letzten Direkteingabe den CHR$-Teil weglassen, dann druckt die Zeile die ASCII-Codes aus, und Sie sehen dann die Leerstellen als Zahl 32.

Diese Resultate habe ich zur besseren Übersicht in Tabelle 8 dargestellt.

Adresse	828	829	830	831	832	833 etc.
Byte Nr.	1	2	3	4	5	6 etc.
Bedeutung	Kennzahl	Low-Byte	High-Byte	Low-Byte	High-Byte	Namen in ASCII-Code
Resultat	1	1	8	41	8	T etc.
bei C 64	1	(2089)		(2049)		
Resultat	1	1	4	41	4	T etc.
bei VC 20		(1065)		(1025)		

Tabelle 8. Tape Header Bytes

Die Kennzahl in Byte »1« können Sie dadurch verändern, daß Sie dem oben verwendeten SAVE-Direktbefehl nach dem langen Namen ein ,1,1 anhängen. Im Ausdruck steht dann die Kennzahl »3«.

Übrigens, wenn Sie in den Speicherzellen 195 und 196 nachschauen, finden Sie dort denselben Wert wie in den Zellen 829/830, so wie die Beschreibung es in der Memory Map erklärt.

Vielleicht fragen Sie jetzt nach dem Nutzen dieser ausführlichen Erklärung. Nun, hauptsächlich kann man damit Programme, die eigentlich wegen LOAD ERROR nicht mehr ladbar sind, retten. Oder aber man kann durch Verändern der Zahlen in den Bytes 2 bis 5 nachträglich die Adressen ändern, in die das Programm geladen wird. Die erste Anwendung werde ich erklären, sobald wir zu den Adressen des Kassetten-Puffers selbst kommen.

Das Problem des LOAD oder SAVE mit geänderten Adressen ist aber zu umfangreich für einen Texteinschub innerhalb dieses Kurses. Es wäre eigentlich einen eigenen kleinen Beitrag wert.

(199) nachzuschauen, was jetzt in der Speicherzelle 199 steht, dann werden Sie Schiffbruch erleiden. Das Betriebssystem setzt den Inhalt der Zelle 199 nach einem »Wagenrücklauf«, hervorgerufen zum Beispiel durch die RETURN-Taste oder nach einem PRINT-Befehl, der nicht mit einem Komma oder Semikolon abgeschlossen ist, sogleich auf 0 zurück. Natürlich erfolgt das auch durch Drücken der CTRL- und RVS-OFF-Taste.

Wir vermeiden die Rücksetzung durch einen Einzeiler:

```
PRINT CHR$(18) "AAA" PEEK
(199)
```

Wir erhalten drei reverse As und als Inhalt der Zelle 199 auch die Zahl 18. Dasselbe Ergebnis erhalten wir durch POKEn einer Zahl größer als 0 in die Zelle 199:

```
POKE 199,4: PRINT"XX" PEEK
(199)
```

Das Ergebnis beweist, daß diese Adresse sehr nützlich sein kann, zumal ihre Abfrage beziehungsweise Beeinflussung auch innerhalb eines Programms erfolgen kann.

Adresse 200 ($C8)

Zeiger auf das Ende der eingegebenen logischen Zeile

Eine echte Zeile faßt beim C 64 maximal 40 Zeichen, beim VC 20 nur 22.

Eine Zeile mit Anweisungen darf beim C 64 insgesamt 80 Zeichen, beim VC 20 sogar 88 Zeichen enthalten. Diese »verlängerte« Programmzeile nennt man »logische Zeile«.

Der Zeiger in Speicherzelle 200 gibt dem Betriebssystem an, auf welcher Position das letzte Zeichen einer eingegebenen logischen Zeile sitzt. Löschen Sie den Bildschirm und geben Sie direkt irgendwo auf dem Bildschirm den Befehl ein:

```
PRINT PEEK(200)
```

Sie erhalten die Zahl der Spalte des letzten Zeichens dieses Direkt-Befehls.

Adresse 201 und 202 ($C9 und $CA)

Zeiger auf Zeilen- und Spaltenposition des letzten Zeichens einer Zeile

Diese beiden Speicherzellen werden bei GET und INPUT verwendet, um die Zeile und Spalte des letzten Zeichens einer eingegebenen Zeile festzustellen. Die Spalten (in Zelle 201 angegeben) zählen von 1 bis 40 (1 bis 22 beim VC 20). Die Zeilen (in Zelle 202 enthalten) zählen dagegen in Paaren von 0 bis 12, identisch mit der bei Zelle 200 erläuterten »logischen« Zweierzeile. Da dies nicht ganz einsichtig ist, gebe ich einen Bildschirmausschnitt wieder (Bild 20), der den Sachverhalt verdeutlichen soll.

```
2345678901234567890123456789012345678900
0
1
1
2
2    PRINT PEEK(201) PEEK(202)
3 2  30
3
4
4
5
5
6      PRINT PEEK(201) PEEK(202)
6 6  32
7
7
8
8
```

Bild 20. Aufbau einer logischen Zeile beim C 64

Der erste Direktbefehl steht in der zweiten Zeile, das letzte Zeichen in der Spalte 30. Der zweite Befehl steht in der ersten Sechserzeile. Das heißt also, daß die Zeilenangabe dieselbe ist, egal, um welchen Teil der logischen Zeile es sich handelt. Das können Sie leicht nachprüfen, indem Sie den ersten Direktbefehl eine Zeile höher schreiben. Das Resultat ist dasselbe.

Die Unterscheidung, um welche der beiden Zeilenteile es sich handelt, wird in den Speicherzellen 217 bis 242 getroffen.

Beim VC 20 sieht der Bildschirmausdruck etwas anders aus (Bild 21), auch die Befehlseingabe habe ich der Zeilenlänge wegen verändert. Interessant ist beim VC 20 allerdings, daß dort trotz der Länge der logischen Zeile auch nur Zeilenpaare verwendet werden, deren Länge natürlich auf 22 Spalten reduziert ist.

```
 2345678901234567890 12
0
1
1
2
2
3
3  PRINTPEEK(201)
4 3
4
5
5
6
6
7    PRINTPEEK(202)
7  19
8
8
```

Bild 21. Aufbau einer logischen Zeile beim VC 20

Adresse 203 ($CB)

Tastencode der gerade gedrückten Taste

Bei der Speicherzelle 145 habe ich beschrieben, wie die Tasten des Computers abgefragt werden. Die dabei für jede der 64 Tasten (mit Ausnahme der RESTORE- und der SHIFT-LOCK-Tasten) entstehende Dualzahl wird in eine Dezimalzahl (0 bis 63) umgewandelt und in der Speicherzelle 203 gespeichert, einige auch in der Zelle 653. Diese Zahl steht auch in Speicherzelle 197, um sie mit der vorher gedrückten Taste vergleichen zu können.

Die Codezahlen jeder Taste lassen sich mit folgendem Programm abfragen:

```
10 PRINT PEEK (203)
20 GOTO 10
```

Nach RUN sehen wir ein laufendes Zahlenband, zuerst mit der Zahl 64. Das ist die Codezahl für »keine Taste gedrückt«. Die X-Taste ergibt 23 (26 beim VC 20), die W-Taste ergibt 9. Auch die Funktionstasten haben ihren Tastencode. F1 ergibt 4 (39 beim VC 20) und so weiter.

Nur die Steuertasten CTRL, SHIFT, und C= (Commodore-Taste) zeigen keine Reaktion. Deren Tastencode steht nämlich in Speicherzelle 653. Den Grund für diesen Separatismus erfahren Sie bei der Besprechung dieser Zelle. Hier ist nur interessant, daß nicht nur jede einzelne dieser drei Tasten einen eigenen Code hat, sondern auch alle machbaren Kombinationen von gleichzeitig gedrückten Steuertasten. Um das zu sehen, ändern Sie bitte die Zeile 10 so ab:

```
10 PRINT PEEK (203),
PEEK(653)
```

Tabelle 9 gibt Ihnen die volle Übersicht. Wenn Sie sich die Mühe machen, die Zahlenreihen der Zelle 203 auf Vollständigkeit zu prüfen, dann werden Sie feststellen, daß vier Zahlen fehlen. Es sind die Werte, die eigentlich den vier Steuertasten CTRL, C=, rechte und linke SHIFT-Taste zugewiesen sind. Aber wie gesagt, sie werden gleich nach 653 umgeleitet, wobei allerdings kein Unterschied mehr zwischen der linken und rechten SHIFT-Taste gemacht wird.

Einige Anwendungsbeispiele der Tastencodes sowie der Kombinationen der drei Steuertasten finden Sie im Texteinschub Nr. 21 »Abfrage der Tastencodes«. Wie schon erwähnt, haben die RESTORE-Taste und die SHIFT-LOCK-Taste keinen eigenen Code.

Die RESTORE-Taste ist überhaupt nicht an die Tastatur-Matrix angeschlossen, sondern ist direkt mit der RESTORE-Leitung des Computers verbunden. Dort löst sie einen sogenannten NMI-Interrupt aus. Die SHIFT-LOCK-Taste ist lediglich eine mechanische Verriegelung der SHIFT-Taste.

Adresse 204 ($CC)

Schalter für Cursor blinken

Ein Wert größer 0 in dieser Speicherzelle schaltet das Blinken des Cursors ab. Diese Abschaltung erfolgt durch das Betriebssystem immer dann, wenn sich Zeichen im Tastaturpuffer befinden und wenn ein Programm ausgeführt wird.

Im folgenden Beispiel einer Eingabe mit dem GET-Befehl, bei dem bekannterweise der Cursor nicht blinkt, wird demonstriert, daß durch POKE 204,0 der Cursor trotzdem blinkt. Das kann für selbstgeschriebene Eingabe-Routinen interessant sein.

```
10 PRINT"JA/NEIN?" ;
20 POKE 204,0
30 GET A$: IF A$=
" "THEN 30
40 PRINT A$
```

Umgekehrt kann man durch POKE 204,1 das Blinken des Cursors abschalten. Es bleibt dabei allerdings dem Zufall überlassen, ob er in der Ein- oder Aus-Phase abgeschaltet wird. Wenn Sie Pech haben, dann bleibt der Cursor bewegungslos stehen.

TASTE	C 64 203	C 64 653	VC 20 203	VC 20 653
nichts	64	0	64	0
F1	4	0	39	0
F3	5	0	47	0
F5	6	0	55	0
F7	3	0	63	0
A	10	0	17	0
B	28	0	35	0
C	20	0	34	0
D	18	0	18	0
E	14	0	49	0
F	21	0	42	0
G	26	0	19	0
H	29	0	43	0
I	33	0	12	0
J	34	0	20	0
K	37	0	44	0
L	42	0	21	0
M	36	0	36	0
N	39	0	28	0
O	38	0	52	0
P	41	0	13	0
Q	62	0	48	0
R	17	0	10	0
S	13	0	41	0
T	22	0	50	0
U	30	0	51	0
V	31	0	27	0
W	9	0	9	0
X	23	0	26	0
Y	25	0	11	0
Z	12	0	33	0
1	56	0	0	0
2	59	0	56	0
3	8	0	1	0
4	11	0	57	0
5	16	0	2	0
6	19	0	58	0
7	24	0	3	0
8	27	0	59	0
9	32	0	4	0
0	35	0	60	0
+	40	0	5	0
−	43	0	61	0
*	49	0	14	0
/	55	0	30	0
=	53	0	46	0
↑	54	0	54	0
←	57	0	8	0
.	44	0	37	0
:	45	0	45	0
,	47	0	29	0
;	50	0	22	0
£	48	0	6	0
@	46	0	53	0
CRSR←	2	0	23	0
CRSR↑	7	0	31	0
DEL	0	0	7	0
HOME	51	0	62	0
STOP	63	0	24	0
RETURN	1	0	15	0
SPACE	60	0	32	0
SHIFT	64	1	64	1
C=	64	2	64	2
CTRL	64	4	64	4
SHIFT und C=	64	3	64	3
SHIFT und CTRL	64	5	64	5
C= und CTRL	64	6	64	6
SHIFT und C= und CTRL	64	7	64	7

Tabelle 9. Tabelle aller Tasten-Codes. Mit den beiden Speicherzellen (203 und 653) lassen sich insgesamt 476 Funktionstasten erzeugen

Adresse 205 ($CD)

Zähler für Blinkfrequenz des Cursors

Das Blinken des Cursors besorgt die Interrupt-Routine. 60mal in jeder Sekunde unterbricht sie den normalen Programmablauf. Während dieser Zeit führt sie mehrere »Haushalt«-Arbeiten durch. So wird hier die Tastatur abgefragt und das Cursorblinken gesteuert.

Dazu wird die Zahl 20 in die Speicherzelle 205 geschrieben und bei jeder Unterbrechung dann um 1 reduziert. Wenn die Zahl in 205 den Wert 0 erreicht hat, wird der Cursor eingeschaltet. Nach Adam Riese erfolgt das also 60/20 = 3mal pro Sekunde. Im Texteinschub Nr. 22 »Cursor-Spiele oder der INPUT-Befehl einmal etwas anders« wird mit diesem Zähler für die Blinkfrequenz experimentiert.

Adresse 206 ($CE)

Bildschirmcode des Zeichens unter dem Cursor

Im Prinzip ist der Cursor nichts anderes als das wiederholte Drucken eines Zeichens in reverser Form, das gerade unter dem Cursor steht. Normalerweise ist dies das Leerzeichen, deshalb sehen wir meistens das ausgefüllte Viereck. Fahren Sie aber mit dem Cursor auf einen Buchstaben, dann erscheint dieser wechselweise normal und revers. In Speicherzelle 206 steht jeweils der Bildschirmcode des Zeichens unter dem Cursor. Geben Sie die folgende Anweisung direkt ein, fahren aber noch vor dem Drücken der RETURN-Taste mit dem Cursor zurück auf eines der Zeichen, zum Beispiel auf ein P:

```
PRINT PEEK(206)
```

Nach RETURN erscheint die Zahl 16. Das ist also der Bildschirmcode des Zeichens, auf dem der Cursor saß, als die RETURN-Taste gedrückt wurde. Sie können das mit allen anderen Zeichen dieser Zeile wiederholen.

Ich kann mir vorstellen, daß eine derartige Abfrage in einem Programm, welches mit dem Bildschirm arbeitet, sinnvoll sein kann. Die Speicherzelle 206 wird allerdings nach jedem Blinken auf den neuesten Stand gebracht.

Texteinschub Nr. 21 Abfrage der Tastencodes oder 476 Funktionstasten

In der Speicherzelle 203 stehen die Tastencodes der gerade gedrückten Taste, insgesamt 64 an der Zahl. Vier davon, die Steuertasten CTRL, C= (Commodore-Taste), linke und rechte SHIFT-Taste, erscheinen allerdings dort nicht, sondern werden sofort in die Speicherzelle 653 umgeleitet. Dort erhalten sie (allerdings in mehrfacher Kombination) insgesamt acht Codewerte. Die Tabelle der Speicherzelle 203 zeigt alle Werte für den C 64 und den VC 20.

Anfänger der Computerei sitzen oft verzweifelt an dem Problem, die Funktionstasten der Commodore-Computer zum Leben zu erwecken. Nun, wir wissen, daß sie nur über die Abfrage ihrer Codewerte eingesetzt werden können.

Als Codewerte werden normalerweise nur die ASCII-Codes genannt.

Die schon erwähnte Tabelle zeigt jedoch, daß die Funktionstasten auch Tastencodes haben. Allerdings gibt uns das nur vier Möglichkeiten, entsprechend der Aufschrift für die ungeraden Funktionstasten-Zahlen. Um auch F2 bis F8 zu erhalten, drücken wir ja immer gleichzeitig die SHIFT-Taste. Das können wir bei der Abfrage der Tastencodes natürlich auch machen, indem wir uns den Inhalt der Zelle 203 und 653 ansehen. Das folgende kleine Programm überprüft, über den Tastaturcode, ob eine der acht Funktionstasten gedrückt wurde.

```
10 A=PEEK(203)
20 B=PEEK(653)
30 IF A=4 AND B=0 THEN PRINT"F1"
40 IF A=5 AND B=0 THEN PRINT"F3"
50 IF A=6 AND B=0 THEN PRINT"F5"
60 IF A=3 AND B=0 THEN PRINT"F7"
70 IF A=4 AND B=1 THEN PRINT"F2"
80 IF A=5 AND B=1 THEN PRINT"F4"
90 IF A=6 AND B=1 THEN PRINT"F6"
100 IF A=3 AND B=1 THEN PRINT"F8"
110 GOTO 10
```

Die Codezahlen gelten für den C 64, für den VC 20 müssen aus der Tabelle die entsprechenden Werte eingesetzt werden.

Wenn Sie sich anschauen, was in der Speicherzelle 653 alles passiert, dann werden Sie sicher sehen, wie willkürlich die Definition der geraden Funktionstasten ist. Statt der Kombination der Funktionstasten mit der SHIFT-Taste können wir genausogut die CTRL-Taste nehmen, oder die Commodore-Taste oder alle zwei oder...oder...!

Mit den acht Codewerten in Zelle 653 (0 bis 7) der acht möglichen Kombinationen der drei Steuertasten kann jede Funktionstaste acht Funktionen haben. Das ergibt insgesamt 32 Funktionstasten und nicht acht, wie die Aufschrift vermuten läßt. Einige davon werden in dem kleinen Demo(nstrations)-Programm eingesetzt. Zweck des Programms soll das Umschalten auf verschiedene Rahmen- und Hintergrundfarben sein. Für den C 64 gilt:

```
10 PRINT CHR$(147)
20 A=PEEK(203)
30 B=PEEK(653)
40 IF A=4 AND B=2 THEN POKE 53280,6:POKE 53281,7
50 IF A=5 AND B=2 THEN POKE 53280,5:POKE 53281,2
60 IF A=6 AND B=2 THEN POKE 53280,1:POKE 53281,1
70 IF A=1 AND B=7 THEN POKE 53280,3:POKE 53281,1
80 GOTO 20
```

Für den VC 20 gilt:

```
10 PRINT CHR$(147)
20 A=PEEK(203)
30 B=PEEK(653)
40 IF A=4 AND B=2 THEN POKE 36879,126
50 IF A=5 AND B=2 THEN POKE 36879,45
60 IF A=6 AND B=2 THEN POKE 36879,25
70 IF A=1 AND B=7 THEN POKE 36879,27
80 GOTO 20
```

Zeile 40 schaltet mit F1 und C= die Farbkombination BLAU/GELB ein.
Zeile 50 schaltet mit F3 und C= die Farbkombination ROT/GRÜN ein.
Zeile 60 schaltet mit F5 und C= die Farbe Weiß ein.

Als Spezialität schaltet Zeile 70 in den Normalzustand zurück, allerdings mit der seltenen Tastenkombination ← (Pfeil links) und alle drei Steuertasten (CTRL, SHIFT, C=) gleichzeitig gedrückt.

Jetzt aber kommt es noch ganz dick!

Ich habe oben gesagt, daß wir nicht acht, sondern 32 Funktionstasten haben. Die Verwendung der vier Funktionstasten in Kombination mit den acht Steuertastencodes in 653 macht es möglich. Dasselbe gilt natürlich für jede andere Taste auch! Zeile 70 im Demo-Programm beweist es.

Da uns insgesamt 60 Tasten zur Verfügung stehen, können wir theoretisch 480 Funktionstasten erzeugen – theoretisch, weil ja auch die STOP-Taste eine gültige Taste ist. Diese Taste steht uns allerdings nur in den Kombinationen mit der SHIFT-Taste zur Verfügung. Ohne SHIFT tut sie ihre Pflicht – sie stoppt. Mit SHIFT aber stoppt sie nicht, so daß wir insgesamt 472 mögliche Kombinationen haben – sicher mehr, als Sie je brauchen werden.

Übrigens, von den Kombinationen sind diejenigen mit der CTRL- oder Commodore-Taste in Spielen oder Anwenderprogramme wie Vizawrite oder Programmierhilfen sehr verbreitet. Ich kann Ihnen nur empfehlen, diese Art der Tastenabfrage ebenfalls zur Steuerung von Programm-Abläufen einzusetzen.

Texteinschub Nr. 22
Cursor-Spiele oder der Input-Befehl einmal etwas anders

Die Speicherzellen 204, 205 und 207 haben alle in einer bestimmten Weise mit dem Cursor zu tun. Da die Details bei jeder dieser Zellen behandelt worden sind, möchte ich hier zusammengefaßt ihren Einsatz an einem kleinen Demo-Programm zeigen. Die Idee zu diesem Programm stammt von Russ Davies (COMPUTE! Publications).

Russ Davies geht von der in vielen Leserbriefen geäußerten Unzufriedenheit mit dem INPUT-Befehl aus, der nicht beliebig lange Zeichenketten zuläßt und sich auch bei versehentlich gedrückter RETURN-Taste schlecht benimmt.

Eine Abhilfe wäre der GET-Befehl, aber der wiederum liefert keinen auffordernden Cursor. In diese Marktlücke springt das folgende kleine Programm, welches die prinzipiellen Anweisungen zeigt für:

- Eingabe langer Zeichenketten mit GET
- blinkender Cursor trotz GET
- veränderbares Blinken des Cursors

```
 10 POKE 211,0
 20 POKE 204,0:POKE 205,5
 30 FOR I=1 TO 40:NEXT
 40 GET A$
 50 IF A$=CHR$(13) THEN 100
 60 PRINT A$;
 70 X$=X$+A$
 80 GOTO 20
100 POKE 204,0:POKE 211,0
120 PRINT X$:PRINT:GOTO 20
```

Zeile 10 verwendet die Speicherzelle 211. Dieser Befehl, auch in Zeile 100, setzt den Cursor auf den Anfang der logischen Zeile zurück. Zeile 20 müßte eigentlich klar sein. Der Wert des POKE-Befehls für 205 ist interessant. Durch ihn kann die Blinkfrequenz des Cursors verändert werden. Bei diesem Programm ergibt der Wert 5 zusammen mit der Warteschleife in Zeile 30 eine mäßige Blinkfrequenz. Der Wert 1 läßt den Cursor eifrig zappeln.

Zeile 30, wie gesagt, dient zur Abstimmung der Cursorfrequenz, die von der Laufzeit der Programmschleife (20 bis 80) abhängt. Trotz des GET-Befehls in Zeile 40 blinkt der Cursor wegen der Flaggen in Speicherzellen 207 und 204.

Zeile 70 baut die Zeichenkette zusammen. Zeile 50 erlaubt ein Drücken der RETURN-Taste, wodurch lediglich die alte Zeichenkette mit der neuen Eingabe zusammengebunden wird. Einen Aussprung aus der Schleife will ich Ihnen selbst überlassen. Im vorliegenden Beispiel geht er nur über die STOP-Taste.

Adresse 207 ($CF)

Flagge für Blinkzustand des Cursors

In dieser Speicherzelle wird festgehalten, in welcher der beiden Blink-Phasen – normal oder revers – der Cursor sich gerade befindet. Eine 0 bedeutet reverses Zeichen, eine 1 bedeutet ein normales Zeichen.

Die Abfrage innerhalb eines Basic-Programms funktioniert nicht. Denn die Interrupt-Routine steuert den Phasenwechsel.

Adresse 208 ($D0)

Flagge für Eingabe von Tastatur oder Bildschirm

Diese Speicherzelle wird von einer Routine des Betriebssystems verwendet, die das jeweils nächste Zeichen in den Arbeitsspeicher holt. Für sie ist wichtig zu wissen, von welchem Eingabegerät dieses Zeichen geholt werden soll.

Wenn in der Zelle 208 eine 0 steht, wird damit die Tastatur als Eingabegerät bestimmt. Das ist der Normalfall, mit dem wir per Tastendruck Zeichen auf den Bildschirm tippen. Sobald aber statt einem Zeichen die RETURN-Taste gedrückt wird, ändert sich der Inhalt der Speicherzelle 208. Die oben genannte Routine überträgt nämlich jetzt den Inhalt der Zelle 213, in welcher die Länge der derzeitigen logischen Zeile steht, nach 208. Dann holt sie das nächste Zeichen, allerdings nicht von der Tastatur, sondern vom Bildschirm, und zwar das erste Zeichen der gerade abgeschlossenen logischen Zeile. Auf diese Weise gelangen die Anweisungen einer Zeile in den Arbeitsspeicher, wo sie im Direkt-Modus sofort ausgeführt, im Programm-Modus aber gespeichert und erst nach RUN ausgeführt werden.

Den Unterschied zwischen »logischer« und »echter« Zeile habe ich in dem Texteinschub Nr. 23 näher beschrieben.

Adresse 209 und 210 ($D1 und $D2)

Zeiger auf den Anfang der Bildschirmzeile, auf welcher der Cursor gerade steht

Dieser Zeiger in Low-/High-Byte-Darstellung zeigt auf die Adresse im Bildschirmspeicher, in welcher diejenige Zeile beginnt, auf der Cursor gerade steht. Das läßt sich leicht nachprüfen durch folgende Programmzeile:

```
10 PRINT CHR$(147) PEEK(209)
PEEK(210)
```

Nach RUN wird erst der Bildschirm gelöscht, der Cursor in die HOME-Position gebracht und dann der Inhalt der beiden Zellen ausgedruckt. Da dies alles in der ersten Zeile passiert, sehen wir als Resultat eine 0 und eine 4. Die beiden Zahlen ergeben zusammen die Adresse, in der die erste Zeile des Bildschirmspeichers beginnt. Erweitern Sie die Zeile 10 um ein Komma und die Low-/High-Byte-Berechnung:

```
10 PRINT CHR$(147) PEEK(209)
PEEK(210), PEEK(209)+256*
PEEK(210)
```

Jetzt sehen wir als Resultat:
0 4 1024

Beim VC 20 erscheinen die der verwendeten Speichererweiterung entsprechenden Zahlen. Wir können durch einen TAB-Befehl den zweiten Teil der PRINT-Anweisung in die nächste Zeile schieben und sehen, was dann herauskommt:

```
20 PRINT PEEK(209) PEEK
(210),TAB(50) PEEK(209)+
256*PEEK(210)
```

Das Resultat ist jetzt:

```
0    4    1024
40   4    1104
```

Einen entsprechenden Zeiger für die Adresse der dazugehörigen Zeile im Farbspeicher werden wir in den Speicherzellen 243 und 244 antreffen. Durch POKEn können wir die Cursorposition leider nicht beeinflussen, aber Abfragen geht, wenn es uns interessiert.

Adresse 211 ($D3)

Position des Cursors innerhalb einer logischen Zeile

Den Inhalt der Speicherzelle 211 könnte man auch die Spaltenposition des Cursors nennen, wenn es sich nicht um die Position in der logischen Zeile handelte (siehe Texteinschub

Nr. 23). Beim C 64 sind daher die Werte von 0 bis 79, beim VC 20 von 0 bis 87 möglich.

Diese Speicherzelle zusammen mit Zelle 214 wird von den Befehlen POS, TAB, SPC und vom Komma innerhalb einer PRINT-Anweisung verwendet, um den Cursor zu positionieren. Das können wir auch. Um den Cursor auf Platz 5 in der Bildschirmzeile 18 zu bringen, geben wir folgende Programmzeile ein:

```
10 POKE 214,17:PRINT:POKE
211,5:PRINT"C 64"
```

Aus innerbetrieblichen Gründen muß der Wert, den wir als Zeile erzielen wollen, um 1 verringert in die Zelle 214 gePOKEt werden. Mit der Zahl 17 wird also der Cursor zuerst auf die Zeile 18 gebracht, dann in Spalte 5, ab der dann das Wort »C 64« gedruckt wird. Auf diese Weise erhalten wir einen Befehl, der in anderen Basic-Formen unter dem Namen PRINT AT sehr verbreitet ist, der bei den kleinen Commodore-Computern aber fehlt.

Der Vorgang dabei besteht darin, daß die Inhalte von 211 und 214 in das X-Register beziehungsweise in das Y-Register des Mikroprozessors gebracht werden. Von dort können die Werte dann von einer Routine des Betriebssystems abgerufen werden. Das klingt alles sehr nach Maschinensprache. Aber wir haben Glück, denn sowohl die beiden Register als auch die besagte Routine sind von Basic aus ansprechbar. Das X-Register steht in Speicherzelle 781, das Y-Register in Speicherzelle 782, die Routine beginnt sowohl beim C 64 als auch beim VC 20 ab der Adresse 58634, wo wir sie mit dem SYS-Befehl starten können.

Für unser Beispiel sieht das dann so aus:

```
10 POKE 781,18:POKE 782,5:
SYS 58634:PRINT"C 64"
```

Wir erhalten dasselbe Ergebnis, nur mit dem Unterschied, daß die Zeile jetzt wirklich die Zeile 18 ist. Mit dieser Methode ist jetzt auch die Zeile 0 erreichbar.

Die Speicherzellen 781 und 782 bieten natürlich noch andere Anwendungen, auf die wir noch kommen werden.

Adresse 212 ($D4)

Flagge für Gänsefuß-Modus

Steht in dieser Speicherzelle eine 0, dann befindet sich der Computer im Gänsefuß-Modus, andere Zahlen bedeuten den Normal-Modus.

Selbst Anfängern ist der Gänsefuß-Modus sehr rasch geläufig, bietet er doch die Möglichkeit, Zeichen mit der PRINT-Anweisung auszudrucken. Genauso bekannt sind aber auch die Tücken der Gänsefüße. Die Cursor-Tasten reagieren nicht wie gewohnt. Auch die Farbumschaltung und andere Steuertasten zeigen nicht die übliche Wirkung, sondern drucken – allzu oft unerwartet – ein reverses Zeichen auf den Bildschirm.

Eingeschaltet wird der Gänsefuß-Modus durch Drücken der geSHIFTeten 2-Taste oder der geSHIFTeten INST/DEL-Taste. Abgeschaltet wird er nach jedem 2., 4., 6., also nach jeder geradzahligen Wiederholung der Gänsefuß-Taste innerhalb einer Zeile. Abgeschaltet wird er auch durch die RETURN-Taste. Das spezielle Verhalten der Steuertasten zwischen Gänsefüßen läßt sich für faszinierende Effekte ausnutzen.

Leider läßt sich der Inhalt der Speicherzelle 212 und damit der Status des Gänsefuß-Modus von Basic aus nicht beeinflussen. Doch in Maschinensprache unter Verwendung der Interrupt-Routine geht es, und einige Vorschläge zum Abschalten des Gänsefuß-Modus per Tastendruck sind schon veröffentlicht worden.

Adresse 213 ($D5)

Länge der Bildschirmzeile

Im Texteinschub 23 »Logische und echte Zeilen« ist der Unterschied zwischen den beiden Zeilentypen beschrieben.

Der Inhalt dieser Speicherzelle entscheidet, wann eine neue logische Zeile begonnen werden muß oder ob die laufende logische Zeile um eine weitere echte Zeile erweitert werden kann. Der Bildschirm-Editor verwendet diese Speicherzelle, um komplette logische Zeilen nach oben zu verschieben. Einige andere Routinen benutzen den Wert der Zelle bei der Rückwärtsüberprüfung einer Zeile, bei der die Endposition der Zeile bekannt sein muß. Schließlich bezieht noch die bereits behandelte Speicherzelle 200 ihren Wert von der Zelle 213.

Adresse 214 ($D6)

Nummer der echten Zeile, auf der sich der Cursor gerade befindet

Diese Speicherzelle ist zusammen mit der Speicherzelle 211 beschrieben.

Adresse 215 ($D7)

Zwischenspeicher für den ASCII-Codewert der zuletzt gedrückten Taste

Bei der Tastaturabfrage werden die Tastencodes (siehe Speicherzelle 203) in ASCII-Codewerte umgewandelt und in den Tastaturpuffer gebracht. Die Speicherzelle 215 dient dabei als Zwischenspeicher. Kassettenoperationen speichern hier auch Prüfsummen ab.

Adresse 216 ($D8)

Flagge für INSERT-Modus

Immer wenn die geSHIFTete INST/DEL-Taste gedrückt wird, um in einer Zeile Platz für ein einzufügendes Zeichen zu schaffen, wird der Inhalt der Speicherzelle 216 um 1 erhöht. Dann wird die Zeile ab dem Freiplatz nach rechts verschoben, der Inhalt der Speicherzelle 213 erhöht und schließlich der entsprechende Wert der Link-Tabelle für Bildschirmzeilen ab Speicherzelle 217 bis 242 verändert.

Bei jedem Tippen eines Zeichens in den freigewordenen Platz wird der Inhalt von 216 wieder um 1 reduziert, bis mit der 0 das Ende des INSERT-Modus angezeigt wird.

Adresse 217 bis 242 ($D9 bis $F2)

Link-Tabellen der Bildschirm-Zeilen

Diese 26 Speicherzellen enthalten Angaben für jede Zeile des Bildschirms. Jedes dieser Bytes hat zwei Funktionen.

Die ersten 4 Bit, also Bit 0 bis 3, geben an, in welchem Speicherblock, man sagt auch »page« dazu, das erste Byte der betreffenden Bildschirmzeile sich befindet. Diese Angabe wird zur Berechnung des Zeigers in der Speicherzelle 209 (siehe dort) verwendet. Sie ist in dieser Form notwendig, da der Bildschirmspeicher beim C 64 überall in den Arbeitsspeicher gelegt werden kann. Um die Position eines Zeichens oder besser gesagt eines Bytes davon im Bildschirmspeicher genau positionieren zu können, braucht das Betriebssystem noch die genaue Lage innerhalb des Speicherblocks. Das Low-Byte dieser Zahl steht in einer Tabelle ab Speicherzelle 60656 (60952 beim VC 20). Das High-Byte wird berechnet, und zwar durch Addition des Wertes der Speicherzelle 648 mit dem Wert der ersten 4 Bit in Tabelle 217 bis 242. Der Wert in Zelle 648 gibt die Anfangsadresse des Bildschirmspeichers an.

Der zweite Teil jedes Bytes in der Tabelle 217 bis 242 hat eine andere Funktion. Wie im Texteinschub 23 beschrieben ist, kann eine logische Zeile aus ein oder zwei (beim VC 20 sogar bis zu 4) echten Zeilen bestehen. Das Betriebssystem braucht daher eine Angabe, welche echten Zeilen zu einer logischen Zeile verbunden sind. Dieses Verbinden heißt auf englisch »link«, daher heißt der Speicherbereich 217 bis 242 »Link-Tabelle«. Diese oberen 4 Bit zeigen mit irgendeinem Wert über 0 an, daß die betreffende echte Zeile die erste oder einzige einer logischen Zeile ist. Sind die 4 Bit alle 0, dann ist sie eine 2., 3. und 4. Zeile der logischen Zeile.

Adresse 243 und 244 ($F3 und $F4)

Zeiger auf Position des Cursors im Farbspeicher

Jedem Platz im Bildschirmspeicher, in dem der Codewert für ein Zeichen steht, entspricht ein Platz im Farbspeicher, in dem der Codewert für die Farbe dieses Zeichens steht.

Das heißt, daß den Bildschirm-Werten der Speicherzellen 209 bis 210 die Farbspeicher-Werte der Zellen 243 bis 244 entsprechen. Dieser Zeiger bestimmt also in der Low-/High-Byte-Darstellung die Adresse im Farbspeicher, ab der die echte Zeile beginnt, auf welcher der Cursor gerade steht.

Adresse 245 und 246 ($F5 und $F6)

Vektor auf die Decodiertabelle für ASCII-Codewerte der Tasten

Bei der Diskussion der Speicherzelle 145 habe ich Ihnen gezeigt, wie das Drücken einer der 64 Tasten entschlüsselt wird.

Ein entschlüsselter Wert wird in Speicherzelle 145 zwischengespeichert und gelangt dann als Tastencode in die Speicherzelle 203. Bei der Besprechung

der Zelle 203 wurden die Codewerte aufgelistet. Ich habe auch darauf hingewiesen, daß die Codes der drei Steuertasten SHIFT, CTRL und COMMODORE (C=) separat in der Zelle 653 stehen.

Diese Tastencodes sind sehr nützlich und vom Basic aus gut verwendbar. Im Verkehr mit anderen Geräten sind sie aber nicht einsetzbar, da sie keiner internationalen Norm entsprechen.

Eine derartige Norm bietet der sogenannte ASCII-Code. Deshalb rechnet, wo notwendig, das Betriebssystem die Tastencodes in den ASCII-Code um.

Dazu stehen im Speicher des Betriebssystems vier Tabellen (Bild 22), die die ASCII-Codewerte enthalten (in Klammern für den VC 20).

ab	60289	(60510):	normale Zeichen
ab	60354	(60575):	Zeichen mit SHIFT
ab	60419	(60640):	Zeichen mit C=
ab	60536	(60835):	Zeichen mit CTRL

Bild 22. Anfangsadressen der 4 Tabellen

Die Umrechnung der Tastencodes in ASCII-Code ist sehr einfach. Der Tastencode wird lediglich zu der Anfangsadresse der entsprechenden Tabelle hinzugezählt. Die Summe ergibt die Adresse in der Tabelle, in der der ASCII-Code für das gedrückte Zeichen steht.

Als Beispiel nehmen wir das normale »G«, sein Tastencode ist 26 (VC 20: 19). Zur Anfangsadresse der normalen Tabelle 60289 (60510) dazugezählt, ergibt das 60315 (60529). Schauen wir in dieser Speicherzelle nach:

```
PRINT PEEK (60315): REM
BEIM C 64
PRINT PEEK (60529): REM
BEIM VC 20
```

In beiden Fällen erhalten wir die Zahl 71. Ein Blick in die ASCII-Tabelle des Handbuchs bestätigt die Richtigkeit.

Der Vektor in den vorliegenden Speicherzellen 245/246 zeigt auf den Anfang der vier Tabellen, und zwar in Abhängigkeit davon, ob und welche der drei Steuertasten zusammen mit einer anderen Taste gedrückt worden ist. Auch das kann ich Ihnen zeigen mit einer Programmzeile, welche ein Zahlenband erzeugt, dessen Zahl durch die Steuertasten verändert wird. Sie werden sehen, es sind die Anfangsadressen der vier Tabellen.

```
10 PRINT PEEK(245)+256*
PEEK(246):GOTO 10
```

Adresse 247 und 248 ($F7 und $F8)

Zeiger auf den Anfang des RS232-Eingabe-Puffers

Immer wenn ein Kanal mit der Geräte-Nummer 2 (User-Port) eröffnet wird, werden am oberen Ende des Arbeitsspeichers zwei Pufferspeicher mit je 256 Byte reserviert (siehe auch die Beschreibung der Speicherzellen 55 bis 56).

Der Zeiger, der in Low-/High-Byte-Darstellung in 247 und 248 steht, zeigt auf die Anfangsadresse desjenigen Pufferspeichers, der die ankommenden Zeichen aufnimmt.

Ein Programm, das den User-Port benutzen will, sollte übrigens immer zuerst die Gerätenummer 2 eröffnen, bevor irgendwelche Variable definiert werden. Dadurch wird vermieden, daß die Puffer-Reservierung eventuelle Variablenwerte überschreibt, die bereits in diesen 512 Byte angesiedelt worden sind.

Adresse 249 und 250 ($F9 und $FA)

Zeiger auf den Anfang des RS232-Ausgabe-Puffers

Dieser Zeiger ist der Zwilling zu dem in den Zellen 247/248 stehenden Zeiger, diesmal aber für den Ausgabe-Puffer.

Adresse 251 bis 254 ($FB bis $FE)

Vier freie Byte für Anwenderprogramme

Diese 4 Byte sind frei, und da sie von Basic nicht gestört beziehungsweise verändert werden, eignen sie sich in idealer Weise für Flaggen, Register oder andere Zwischenspeicher.

Adresse 255 ($FF)

Zwischenspeicher für Daten bei der Umwandlung von Gleitkomma-Zahlen in ASCII-Werte

Der Titel dieser Speicherzelle sagt schon alles.

Jetzt haben wir eine erste Etappe unserer Wanderung durch die Speicherlandschaft hinter uns, nämlich den Bereich von 0 bis 255. Man nennt ihn »Zero-Page«, und er hat in der Maschinensprache-Programmierung beziehungsweise in der Adressierung eine spezielle Bedeutung. Für Basic-Programmierer ist diese Seiteneinteilung bedeutungslos, und wir werden deshalb unbeirrt als nächstes mit einem größeren Speicherblock, nämlich von 256 bis 511 weitermachen.

Texteinschub Nr. 23
Logische und echte Zeilen

Der Bildschirm des C 64 besteht aus 25 Zeilen. Jede davon enthält 40 Stellen. Der VC 20 hat 23 Zeilen mit je 22 Stellen. Diese Zeilen des Bildschirms werden »echte Zeilen« genannt.

Der Begriff »Zeile« kommt auch beim Programmieren vor. Dort bedeutet er die Reihe der Anweisungen und Befehle, die hinter einer Zeilennummer zusammengefaßt sind. Diese Programmzeilen werden »logische Zeilen« genannt.

Wozu dient dieser Unterschied?

Der Grund ist ganz einfach der, daß sehr oft eine Programmzeile mehr Zeichen enthält, als in eine (echte) Zeile des Bildschirms hineinpassen. Besonders Texte innerhalb einer PRINT-Anweisung sind häufig viel länger als 40 Zeichen, erst recht als die 22 Zeichen beim VC 20.

Deshalb läßt es das Betriebssystem des Computers zu, daß in einer (logischen) Programmzeile mehr als 40 (22) Zeichen stehen.

Beim C 64 kann eine logische Zeile aus zwei echten Zeilen bestehen mit einer maximalen Zeichenzahl von 80.

Der VC 20 erlaubt in einer logischen Zeile maximal 88 Zeichen und braucht dazu 4 echte Zeilen.

Dieses Zusammenfügen von mehreren echten Zeilen zu einer logischen Zeile hat natürlich Konsequenzen. Am meisten davon betroffen ist der Editor, das ist der Teil des Betriebssystems, der uns erlaubt, auf dem Bildschirm zu arbeiten, zu ändern, zu löschen und fertige Zeilen mit RETURN einzugeben.

Ich will Ihnen nur ein paar Beispiele nennen, die Sie sicher kennen, um Ihnen den Zusammenhang zu zeigen.

Wenn Sie ein Programm LISTen und anschließend in einer Programmzeile weitere Befehle anhängen, die über die Länge der echten Zeile hinausgehen, brauchen Sie selbst keinen Platz dafür schaffen. Das macht der Editor automatisch, indem er alle folgenden Zeilen auf dem Bildschirm nach unten schiebt. Für die Einsteiger unter Ihnen führe ich das vor. Geben Sie bitte die folgenden drei Zeilen ein (mit RETURN abschließen):

```
10 PRINT "A"
20 PRINT "B"
30 PRINT "C"
```

Jetzt fahren Sie mit dem Cursor auf die Zeile 20 und schreiben statt einem B so viele davon, bis Sie damit in die nächste Zeile kommen.

Sie werden sehen, daß dabei die Zeile 30 von selbst eins weiter nach unten rutscht. Erst wenn Sie in die 3. Zeile (5. Zeile beim VC 20) kommen, wird die Zeile 30 überschrieben. Sie sind dann über das Ende der logischen Zeile geraten. Eine überfüllte logische Zeile kann nicht eingegeben werden, die RETURN-Taste wird mit SYNTAX ERROR quittiert beziehungsweise abgewiesen.

Es gibt allerdings eine Methode, wie Sie in einem Listing eines Programms Programmzeilen, also logische Zeilen mit mehr als 80 (88) Zeichen erhalten können.

Fast alle Basic-Befehle können in abgekürzter Form eingetippt werden, am häufigsten wird sicher das Fragezeichen ? anstelle von PRINT eingesetzt. Sie können nun eine logische Zeile mit Abkürzungen und durch Weglassen von allen Zwischenräumen bis zum erlaubten Maximum füllen. Natürlich wird diese volle Zeile nach RETURN akzeptiert. Beim ausLISTen aber schreibt der Editor alle Befehle, auch die abgekürzten, in voller Länge aus und fügt nach der Zeilennummer einen Zwischenraum ein. Eine solche Zeile hat dann mehr echte Zeilen als erlaubt, intern aber hat sie die richtige Länge.

Einen Nachteil hat diese Komprimierung doch. In einer solchen »überlangen« logischen Zeile kann nicht mehr korrigiert

werden, es sei denn, sie wird vorher auf eine erlaubte Länge reduziert.

Abschließend möchte ich noch kurz erwähnen, daß zur Steuerung dieser Zusammenhänge zwischen echten und logischen Zeilen die Speicherzellen 200, 201, 202, 209, 211, 213, 214, 217 bis 242 und 658 eine entsprechende Rolle spielen, die im einzelnen in der Memory Map beschrieben ist.

Adresse 256 bis 266 ($100 bis $10A)

Arbeitsspeicher für Umwandlung von Gleitkomma-Zahlen in ASCII-Werte, auch FAC (Fließkomma-Akku) genannt

Diese 11 Byte werden von einer Routine des Betriebssystems verwendet, um Werte zwischenzuspeichern, die bei der Umwandlung von Gleitkomma-Zahlen in ASCII-Werte oder in Werte der Funktion TI$ anfallen. Eine andere Routine verwendet den Bereich, um Zeichenketten (Strings) zu untersuchen.

Adresse 256 bis 318 ($100 bis $13E)

Arbeitsspeicher für Fehler bei der Eingabe vom Band

Alle Daten, die auf Band gespeichert werden, stehen dort doppelt in zwei identischen Blöcken hintereinander. Beim Laden in den Computer werden beide Blöcke miteinander verglichen, um Fehler zu finden und, wo möglich, sie zu korrigieren.

In diesem Bereich, der übrigens auch bei der Speicherzelle 256 anfängt, aber 63 Byte in Anspruch nimmt, werden beim Laden Angaben gespeichert, aus denen das Betriebssystem erkennen kann, welche Bytes fehlerhaft sind.

Adresse 319 bis 511 ($13F bis $1FF)

Stapelspeicher (Stack) des Mikroprozessors

Die Funktionsweise eines Stapelspeichers, auf englisch »stack«, ist im Texteinschub Nr. 24 erklärt.

Der Stapelspeicher hat prinzipiell die Aufgabe, bei allen Sprüngen oder Unterbrechungen innerhalb eines normalen Programmablaufs alle Adressen und Daten so zu speichern, daß am Ende der Unterbrechung das Programm wieder fortgesetzt werden kann.

Derartige Unterbrechungen und Sprünge treten in Basic bei den Befehlen GOSUB-RETURN und FOR-NEXT auf, genauso wie bei vielen Routinen des Betriebssystems. In Maschinensprache gibt es dafür sogar eigene Befehle. Heimo Ponnath hat sie alle in seinem Assemblerkurs (Folge 7 und 8 im 64'er, Ausgabe 2/85 und 3/85) sehr ausführlich beschrieben.

Da uns hier Basic mehr interessiert, gebe ich Ihnen nur kurz an, was im Stapel gespeichert wird, da der Stapelspeicher nur in Maschinensprache manipuliert werden kann.

Jeder FOR-TO-NEXT-Befehl belegt 18 Byte im Stapelspeicher.

Im ersten Byte steht als Kennung die Zahl 129. Byte 2 und 3 enthalten in Low/High-Byte-Darstellung einen Zeiger auf die Adresse, in der die durch das FOR definierte Schleifen-Variable (zum Beispiel K in FOR K=0 TO 3) gespeichert ist. Die nächsten 5 Byte sind für den Gleitkommawert von STEP reserviert, das Byte danach für das Vorzeichen von STEP. Danach folgt der Gleitkommawert von TO mit 5 Byte und in zwei weiteren Byte die Nummer derjenigen Zeile, auf die nach dem NEXT zurückgesprungen wird. In den letzten beiden Bytes schließlich steht ein Zeiger auf der Adresse, in der das nächste Zeichen steht, welches nach Beendigung der FOR-TO-NEXT-Schleife gelesen werden muß.

Ein GOSUB-Befehl belegt 5 Byte im Stapelspeicher. Byte 1 enthält die Kennzahl 141. Ihr folgen zwei Byte für die Nummer der Zeile, auf die nach RETURN zurückgesprungen wird. Die letzten beiden Byte enthalten wieder einen Zeiger auf die Adresse, in der das nächste Zeichen steht, mit dem nach RETURN das Programm fortgesetzt wird.

Der Basic-Befehl DEF zur freien Definition von Funktionen belegt ebenfalls 5 Byte im Stapelspeicher. Ihre Verteilung ist dieselbe wie von GOSUB, mit dem einzigen Unterschied, daß statt der ersten Kennzahl irgendein anderer Wert verwendet wird, der aber keine Bedeutung hat.

Wenn so viele FOR...NEXT-Schleifen oder GOSUB-Sprünge gleichzeitig im Programm vorkommen, daß der Stapelspeicher voll wird, steigt das Programm mit OUT OF MEMORY aus.

Adresse 512 bis 600 ($200 bis $258)

Eingabespeicher von Basic

Wenn Sie Zeichen, zum Beispiel einen Befehl oder eine Programmzeile, eingeben und mit der RETURN-Taste abschließen, werden diese Zeichen in diesen Speicherbereich von 512 bis 600 gebracht. Seine Länge von 89 Byte entspricht der Länge einer logischen Zeile des VC 20 (88 Zeichen) plus einer Abschluß-Null. Die logische Zeilenlänge des C 64 von 80 Zeichen füllt den Speicherbereich nicht ganz aus, aber das Betriebssystem des C 64 ist gegenüber dem des VC 20 nicht geändert worden.

Nach RETURN sucht der Computer diesen Eingabespeicher nach Gänsefüßen, Komma und nach der Zahl für Zeilenende ab. Dann wandelt der Computer die gespeicherten Zeichen in für ihn lesbare Zahlen (Token und ASCII-Werte) um und fügt am Anhang die Zeilennummer und die Anschluß-Adresse (Link) der nächsten Zeile, am Ende die Abschluß-Null hinzu. Wenn eine Zeilennummer vorhanden ist, kommt alles in den Programmspeicher. Fehlt sie jedoch, dann wird die ganze Anweisung sofort ausgeführt (Direktmodus).

Eine detaillierte Beschreibung dieses Eingabe- und Umwandlungsvorganges gab Christoph Sauer im 3. Teil seines Kurses »Der gläserne VC 20« im 64'er, Ausgabe 11/84 ab Seite 126.

Dieser Speicherbereich wird auch von den Befehlen INPUT und GET benutzt, um die Eingabedaten aufzunehmen. Das erklärt übrigens, warum diese beiden Befehle nur innerhalb einer Programmzeile und nicht im Direktmodus verwendet werden können. Sie verwenden ja denselben Speicherplatz, der vom Direkt-(Eingabe-)Modus verwendet wird.

Es erklärt außerdem, warum eine von INPUT geforderte Eingabe maximal 88 Zeichen lang sein darf.

Adresse 601 bis 630 ($259 bis $276)

Tabellen für File-Nummern, Geräte-Nummern und Sekundär-Adressen von eröffneten Dateien

Bei der Besprechung der Speicherzelle 152 habe ich diesen Speicherbereich bereits erwähnt. Ich habe damals gesagt und gezeigt, daß die Zelle 152 über die Anzahl der eröffneten Dateien (Files) Buch führt, die Tabellen in 601 bis 630 dagegen darüber, welche File-Nummern, Geräte-Nummern und Sekundär-Adressen jeder eröffneten Datei zugeordnet sind. Wer sich nochmals über diese Begriffe orientieren will, den verweise ich auf die Beschreibung im Texteinschub Nr. 19.

Der Speicherbereich von 601 bis 630 ist in drei Blöcke unterteilt.

601 bis 610	Tabelle der File-Nummern
611 bis 620	Tabelle der Geräte-Nummern
621 bis 630	Tabelle der Sekundär-Adressen

In jeder Tabelle können also maximal 10 Byte stehen. Sie haben folgende Zusammensetzung:

Die drei Angaben über eine eröffnete Datei stehen in den Tabellen jeweils am gleichen Platz. Wenn also die Datei Nummer 5 als dritte Datei eröffnet worden ist, steht eine 5 in Zelle 603, ihre Gerätenummer in Zelle 613, die Sekundär-Adresse entsprechend in Zelle 623.

Immer wenn eine neue Datei eröffnet wird, kommen diese Angaben auf die nächsten Plätze der Tabellen, und der Inhalt der Speicherzelle 152 wird um 1 erhöht. Wird dagegen eine Datei geschlossen, dann rücken alle Angaben dahinter um eine Stelle zurück.

In diesen Tabellen kann nachgesehen werden, mit welchen Parametern Dateien eröffnet worden sind. Eine sehr interessante Anwendung, die Eintragung in den Tabellen durch POKE zu verändern, wurde von Rügheimer und Spanik veröffentlicht, welche ich hier zitieren möchte.

Eine Änderung der Filenummern in Tabelle 601 bis 610 ist nicht empfehlenswert, um Verwechslungen zu vermeiden.

Mit dem folgenden kleinen Programm, welches bei einer eröffneten Datei die Geräte-

Nummer in Tabelle 611 bis 620 ändert, kann zwischen einem Drucker mit Geräte-Nummer 4 und einem Plotter mit Geräte-Nummer 6 umgeschaltet werden.

```
10 OPEN 4,4,0
20 POKE 611,6: PRINT#4,
"PLOTTER"
30 POKE 611,4: PRINT#4,
"DRUCKER"
```

Ähnliches ist mit den Sekundär-Adressen möglich. Dabei muß man allerdings wissen, daß die Sekundär-Adressen nicht so wie sie sind in der Tabelle 621 bis 630 gespeichert werden, sondern mit »OR 96« verknüpft. Dasselbe müssen wir auch machen:

```
10 OPEN 3,4,0
20 POKE 621,0 OR 96:
PRINT#3, "GRAFIKMODUS"
30 POKE 621,7 OR 96:
PRINT#3, "TEXTMODUS"
```

Mit dieser Methode können Sie sich das Öffnen und Schließen vieler Dateien ersparen.

Adresse 631 bis 640 ($277 bis $280)

Tastaturpuffer

Bei der Behandlung der Speicherzelle 203 habe ich die Codezahlen beschrieben, die beim Drücken einer der 64 Tasten erzeugt werden. Bei den Speicherzellen 245 und 246 haben wir gesehen, wie aus diesen Tastencodes der ASCII-Code für die verschiedenen Zeichen einer Taste umgerechnet wird.

Hier nun im Tastaturpuffer landen diese umgerechneten ASCII-Werte. Wenn Sie den Kurs schon länger verfolgen, ist Ihnen das auch nicht neu, denn ich habe die Wirkungsweise des Tastaturpuffers bei der »Dynamischen Tastenabfrage« (Texteinschub Nr. 15) sozusagen im Vorgriff, ausführlich erklärt.

Zur Erinnerung sei gesagt, daß im Tastaturpuffer alle Zeichen zwischengespeichert werden, die während eines Programmlaufes eingegeben und nicht sofort vom Betriebssystem verarbeitet werden können. Sobald der Computer sich im Eingabe-Modus befindet — nach Programmende oder bei INPUT- und GET-Befehlen, werden die Zeichen in der Reihenfolge ihrer Eingabe herausgeholt und verwendet.

Der Tastaturpuffer ist 10 Byte lang. In Speicherzelle 198 steht, wieviele Zeichen sich im Puffer befinden.

Als Ergänzung zu den Beispielen der dynamischen Tastenabfrage zeige ich Ihnen im Texteinschub Nr. 25 »Programme, die sich selbst verändern« noch ein paar andere Anwendungen.

Adresse 641 und 642 ($281 und $282)

Zeiger auf den Anfang des Programmspeichers

Wenn der Computer eingeschaltet wird oder wenn mit einer Reset-Taste beziehungsweise mit SYS58260 (VC 20: SYS 58232) ein Kaltstart ausgelöst wird, setzt das Betriebssystem diesen Zeiger auf die Adresse des ersten freien RAM-Speicherplatzes.

Beim C 64 ist dies die Adresse 2048. Beim VC 20 hängt sie von der Speichererweiterung ab; ohne Erweiterung ist es 4096, mit einer 3-KByte-Erweiterung dagegen 1024, mit 8 KByte oder mehr ist die Adresse 4608.

Dieser Zeiger wird vom Basic-Übersetzer in die Speicherzelle 43 übernommen und nur von dort weiterverwendet.

Adresse 643 und 644 ($283 und $284)

Zeiger auf das Ende des Programmspeichers

Dieser Zeiger ist der Zwilling zu dem anderen Zeiger in 641 und 642. Er wird vom Betriebssystem auf die Adresse gesetzt, welche beim Kaltstart beziehungsweise der dabei durchgeführten Prüfung des Speichers den letzten verfügbaren RAM-Speicherplatz angibt. Beim C 64 ist diese Adresse normalerweise 40960 ($A000), beim VC 20 ohne Erweiterung 7680.

Dieser Zeiger wird vom Basic-Übersetzer in die Speicherzelle 55 übernommen.

Adresse 645 ($285)

Flagge für Ein- und Ausschalten der IEEE-488-Karte

Diese Speicherzelle ist etwas mysteriös. Sie kommt im ganzen Betriebssystem nur ein einziges Mal zum Einsatz, und zwar als Flagge beim Betrieb der sogenannten IEEE-488-Interface-Karte. Wenn diese Flagge gesetzt ist, wartet der Computer 64 Millisekunden lang, ob er von einem angeschlossenen Gerät angesprochen wird. Wenn kein Signal kommt, gibt er ein Fehlersignal aus.

Zahlen in der Zelle 645, die kleiner als 128 sind, bedeuten Flagge gesetzt, größer als 128 löschen sie die Flagge.

Texteinschub Nr. 24
Der Stapelspeicher

Der normale Speicher, mit dem wir es immer zu tun haben, in den wir Zahlen hineinPOKEn oder herausPEEKen, ist aufgebaut wie eine Häuserreihe, in der jedes Haus seine eigene Adresse hat. Wer etwas in einem bestimmten Haus abliefern oder aus ihm abholen will, muß seine genaue Adresse kennen.

Dieses Speicherprinzip heißt RAM, das ist die Abkürzung für »Random Access Memory« oder auf deutsch »Speicher mit wahlfreiem Zugriff«.

Der Stapelspeicher funktioniert anders. Jetzt werden die Häuser der Reihe nach aufgesucht, wie mit einer Postwurfsendung oder durch die Müllabfuhr. Ein besserer Vergleich ist der Aktenstapel. Die erste Akte wird auf den Tisch gelegt, alle nachfolgenden kommen obendrauf.

Beim Aktenstapel sieht man eine weitere Eigenschaft. Wenn er nämlich abgearbeitet wird, kommt die Akte als erste an die Reihe, die als letzte auf den Stapel gelegt worden ist.

Dieses Speicherprinzip heißt »LIFO«; das ist eine Abkürzung aus dem Englischen und heißt »Last In First Out«.

Der Benutzer eines Stapelspeichers braucht sich nicht mehr um Adressen zu kümmern. Er kennt nur noch den Platz, wo der Speicher Daten annimmt beziehungsweise abgibt. Was der Benutzer sich allerdings merken muß, ist die Reihenfolge seiner Ein- und Ausgaben.

Stapelspeicher werden von mehreren Programmiersprachen verwendet; am bekanntesten ist wohl FORTH, auch die HP-(Hewlett-Packard-)Taschenrechner arbeiten nach diesem Prinzip.

Unsere Computer verwenden das Prinzip des Stapelspeichers nur bei Programmschleifen, Unterprogramm-Sprüngen, ja überhaupt bei jeder Unterbrechung des normalen Programmablaufes. Das Problem dabei ist nämlich, sich alle Adressen und Angaben des Programms so zu merken, daß nach dem Ende der Unterbrechung das alte Programm lückenlos fortgesetzt werden kann.

Der Stapel belegt den Speicherbereich von 256 bis 511. Dieser Bereich unterscheidet sich in seinem Aufbau natürlich überhaupt nicht von den anderen Speicherbereichen. Es wäre auch viel zu aufwendig, alle Daten des Stapelspeichers bei jeder Ein- und Ausgabe rumzuschieben. Diese Aufgabe besorgt ein Register des Mikroprozessors, das »Stapelzeiger« genannt wird und das wie ein Zähler arbeitet.

Am Anfang steht in diesem Stapelzeiger die oberste Adresse 511. Bei jeder Eingabe wird die Zahl um 1 erniedrigt, bei jeder Ausgabe um 1 erhöht. Da der Stapelzeiger, wie jedes andere Register auch, eine Länge von 8 Bit hat, kann er nur die Dualzahlen von 00000000 bis 11111111 darstellen. Um daraus 256 beziehungsweise 511 zu formen, stellt der Mikroprozessor sozusagen fest verdrahtet ein neuntes Bit, immer auf 1, der Zahl voran.

Texteinschub Nr. 25
Programme, die sich selbst verändern

Der Speicherbereich von 631 bis 640 beherbergt den Tastaturpuffer. Schon im Texteinschub Nr. 15 habe ich Ihnen Anwendungen gezeigt, die den Tastaturpuffer einsetzen und in der Literatur unter dem Namen »Dynamische Tastenabfrage« bekannt sind.

Hinter diesem natürlich aus dem Englischen übersetzten Begriff verbirgt sich die Möglichkeit, innerhalb eines Programms bestimmte Werte in den Puffer zu speichern, die dort so lange aufgehoben bleiben, bis das Programm – aus welchen Gründen auch immer – beendet wird. Dann erst werden die Werte hervorgeholt. Wenn es Zeichen sind, dann erscheinen sie auf dem Bild-

schirm. Wenn es Steuertasten sind (zum Beispiel RETURN), werden sie ausgeführt.

Ich will hier nicht die schon gezeigten Anwendungen wiederholen, sondern sie lediglich um ein paar weitere Tricks ergänzen.

Zeile in ein Programm einfügen

Die folgende Methode ermöglicht die Veränderung eines Programms durch sich selbst. Genauer gesagt, man kann damit Programmzeilen einfügen. Nehmen wir an, Sie haben folgendes Teilprogramm:

```
150   PRINT"BITTE TASTE DRUECKEN"
160   GET A$:IF A$=""THEN 160
170   IF A$ <> "E" THEN 150
180   END
...
...
500   PRINT "NAECHSTER TEIL"
```

Die ersten vier Zeilen warten so lange, bis die E-Taste gedrückt wird, dann bleibt das Programm mit READY stehen.

Wir wollen nun eine zusätzliche Zeile 165 durch das Programm einfügen lassen, mit der nicht beendet, sondern durch Drücken der Q-Taste auf die Zeile 500 gesprungen wird. Das erreichen wir durch folgende Zeilen:

```
172   PRINT CHR$(147)
174   PRINT "165 IF A$=CHR$(81) THEN 500"
176   PRINT "GOTO 150"
178   POKE 631,19:POKE 632,13:POKE 633,13:
      POKE 634,13:POKE 198,4
```

Wenn jetzt die E-Taste gedrückt wird, löscht Zeile 172 den Bildschirm, Zeile 174 druckt in die zweite Bildschirmzeile die neue Programmzeile 165 und Zeile 176 darunter ohne Zeilennummer den Direktbefehl GOTO 150. In den Tastaturpuffer werden nacheinander die Werte für HOME und dreimal RETURN eingegeben und die Anzahl der Zeichen im Puffer auf 4 begrenzt.

Sobald nun das Programm den Befehl END in Zeile 180 erreicht, werden die Werte im Puffer ausgeführt, das heißt der Cursor geht nach HOME, das erste RETURN setzt ihn ohne Wirkung eine Zeile tiefer, wo er auf der neu ausgedruckten Zeile 165 steht. Das zweite RETURN gibt diese Zeile ein, das dritte RETURN führt den Direktbefehl GOTO 150 aus, wodurch das Programm weiterläuft, so wie vorher. Der einzige Unterschied ist nur, daß jetzt eine neue Zeile im Programm steht, nämlich die Zeile 165.

Prüfen Sie es mit STOP und LIST.

Zeile verändern oder löschen

Diese Methode habe ich von S. Leemon übernommen. Sie funktioniert im Prinzip genauso wie das Einfügen von Zeilen. Die Programmzeilen sind durch REMs erklärt. Probieren Sie es aus.

```
10   REM -- DIESE ZEILE WIRD GELÖSCHT
20   PRINT CHR$(147):PRINT:PRINT
30   PRINT "60 LIST"
40   PRINT "10"
50   PRINT "GOTO 70" CHR$(19)
60   FOR I=631 TO 633:POKE I,13:NEXT: POKE 198,3:END
70   REM -- DIESE ZEILE WIRD ERSETZT
```

Diese Verfahren, Zeilen eines Programms während des Laufs eines Programms zu ändern, haben sicher für einfache Programme keine große Bedeutung. Ich bin aber fest davon überzeugt, daß sie bei Programmen, die selbst lernen können, eine wichtige Rolle spielen. Nur habe ich hier jetzt noch kein gutes Demonstrationsbeispiel.

Einfügen einer Funktion

Sie alle kennen die Möglichkeit, Funktionen mit dem Basic-Befehl DEF FN.. selbst zu definieren. Vielleicht haben Sie auch schon einmal frustriert die Versuche abgebrochen, eine Funktion während des Programmablaufs per INPUT eingeben zu können. Das funktioniert nämlich nicht.

Mit dem Tastaturpuffer aber geht es, und zwar sehr elegant. Der vergebliche Versuch sieht so aus:

```
150   PRINT "FORMEL EINGEBEN"
160   INPUT "F(X)=";A$
...
250   DEF FNA(X)=A$
260   PRINT FNA(9)
270   GOTO 150
```

Der STRING A$, in Zeile 160 eingegeben, wird von Zeile 250 nicht übernommen. Zeile 250 können Sie so nicht verwenden, also löschen Sie sie bitte.

Sie können diese Zeile aber vom Programm einfügen lassen:

```
170   PRINT CHR$(147):PRINT:PRINT
180   PRINT "250 DEF FNA(X)=";A$
190   PRINT "GOTO 240"
200   PRINT CHR$(19)
210   POKE 631,13: POKE 632,13: POKE 198,2: END
...
...
240   REM -- FORMELBERECHNUNG --
```

Wenn Sie nicht genau nachvollziehen können, was da vorgeht, empfehle ich Ihnen, an das Ende der Zeile 200 noch ein END anzuhängen. Dann bleibt das Programm dort stehen, Sie sehen, wo der Cursor steht und können dann die Wirkung der Zeile 210 in Ruhe überlegen.

Eine weitere interessante Anwendung habe ich bei Russ Davies gefunden. Sie ermöglicht, Routinen des Betriebssystems oder Basic-Übersetzers aus dem Speicher auszulesen und als DATA-Anweisungen in Ihr Basic-Programm einzufügen.

Einfügen von DATA-Zeilen

Die Aufgabe besteht darin, aus einem auszuwählenden Speicherbereich – dort, wo die in Frage kommende Routine sitzt – den Maschinencode herauszuPEEKen und mit der inzwischen bekannten Methode in eine DATA-Zeile einzufügen. Ich gehe schrittweise vor:

```
20   PRINT CHR$(147): PRINT: PRINT
30   PRINT "5555 DATA";
50   FOR A=0 TO 3
60   PRINT PEEK(A)", ";:NEXT
70   PRINT CHR$(19)
80   POKE 631, 13: POKE 632,13: POKE 198,2: END
```

Mit Zeilen 20 und 30 wird der DATA-Zeile die Zeilennummer 5555 gegeben und in die 3. Bildschirmzeile geschrieben. Zeile 50 liest die Speicherzellen 0 bis 3 aus und druckt sie, mit einem Komma versehen, dahinter. Zeilen 70 und 80 geben diese Zeile in gewohnter Manier in das Programm ein.

Ganz richtig ist das noch nicht, da nach LIST die Zeile 5555 zu viele Zwischenräume hat:

```
5555 DATA 47 , 55 , 0 , 170 ,
```

Zur Korrektur müssen wir Zeile 60 erweitern:

```
60 PRINT MID$(STR$(PEEK(A)),2)",";:NEXT
```

STR$() wandelt den durch PEEK ausgelesenen Zahlenwert in einen String um. MID$(...,2) schneidet die Leerstellen weg.

Mr. Davies hat daraus eine komfortable kleine Routine gebaut, die den auszulesenden Speicherbereich und die gewünschte Zeilennummer der ersten DATA-Zeile abfragt, dafür sorgt, daß eine DATA-Zeile nicht länger als 16 Zeichen wird, in diesem Fall die nächste Zeilennummer um 10 erhöht und sogar das letzte Komma in jeder DATA-Zeile eliminiert.

Dieser ganze Komfort hat natürlich nichts mit dem Tastaturpuffer zu tun. Dieser Teil bleibt unverändert. Aber ich finde dieses Programm so durchdacht, daß ich es Ihnen als ein Beispiel guter Programmierung angeben will.

```
10   INPUT "STARTADRESE";A: INPUT "ENDADRESE";E:
     INPUT "ZEILE";Z
20   PRINT CHR$(147)
30   PRINT Z "DATA";
40   IF A> E THEN END
50   FOR A=A TO A+15 + (E<A+15)*(A+15-E)
60   PRINT MID$(STR$(PEEK(A)),2)",";:NEXT
70   PRINT"[CRSR-links]":PRINT "A="A":
     E="E": Z="Z+10": GOTO 20";
80   POKE 631, 19: POKE 632,13: POKE 633,13:
     POKE 634,13: POKE 198,4: END
```

Adresse 646 ($286)

Aktuelle Farbe der Zeichen (Vordergrundfarbe)

Um ein bestimmtes Zeichen auf den Bildschirm zu drucken, muß vom Betriebssystem erstens der Bildschirmcode des Zeichens in den Bildschirmspeicher und zweitens der Codewert der gewünschten Farbe in den Farbspeicher gebracht werden.

In der Speicherzelle 646 steht immer der Codewert derjenigen Farbe, die gerade eingestellt ist. Immer wenn ein PRINT-Befehl gegeben wird, holt das Betriebssystem den Farbwert aus der Zelle 646 und bringt ihn in den Farbspeicher, und zwar an den entsprechenden Platz, wo gerade gePRINTet werden soll. Der Codewert in der Zelle 646 kann auf drei Arten eingestellt werden:

- Drücken der CTRL-Taste gleichzeitig mit einer der Farbtasten 1 bis 8. Beim C 64 kommen noch weitere acht Farben dazu durch Drücken der Commodore-Taste anstelle der CTRL-Taste.
- PRINT-Befehl gefolgt vom ASCII-Codewert der Farbe innerhalb von Gänsefüßen.
- POKEn der Farbcodes 0 bis 7 (beim C 64 0 bis 15) direkt in die Speicherzelle.

Innerhalb eines Programms ist das POKEn in Zelle 646 wohl die eleganteste Methode (Tabelle 10).

Als Beispiel möge dieses kleine Programm dienen:

```
10 FOR X=0 TO 7
20 POKE 646,X
30 PRINT "A";
40 NEXT X
50 GOTO 10
```

Wer mehr über Vordergrund- und Hintergrundfarben erfahren will, der lese den Texteinschub Nr. 26 »Bunte Zeichen und bunter Hintergrund«.

FARBE	CODE	ASCII	TASTEN	FARBE	CODE	ASCII	TASTEN
schwarz	0	144	CTRL+1	orange	8	129	CBM+1
weiß	1	5	CTRL+2	braun	9	149	CBM+2
rot	2	28	CTRL+3	hellrot	10	150	CBM+3
lila	3	159	CTRL+4	dunkelgrau	11	151	CBM+4
purpur	4	156	CTRL+5	mittelgrau	12	152	CBM+5
grün	5	30	CTRL+6	hellgrün	13	153	CBM+6
blau	6	31	CTRL+7	hellblau	14	154	CBM+7
gelb	8	158	CTRL+8	hellgrau	15	155	CBM+8

Tabelle 10. Tabelle der Farben und ihrer Codes beziehungsweise Tasten

Adresse 647 ($287)

Zeichenfarbe unter dem Cursor

Das Blinken des Cursors wird dadurch erzeugt, daß das Zeichen auf der Stelle des Bildschirms, auf der er gerade steht (meistens ist es eine Leerstelle), dauernd von »normal« auf »revers« (oder »invertiert«) und zurück geschaltet wird. Die reverse Darstellung benutzt dabei die Farbe des Zeichens.

Genauso, wie sich der Computer in der Speicherzelle 206 das Zeichen merkt, mit dem er gerade blinkt, um beim Weiterwandern dieses Zeichen in seiner »normalen« Form auf dem Bildschirm zurückzulassen, merkt er sich die Farbe dieses Zeichens in der Speicherzelle 647.

Adresse 648 ($288)

Beginn des Bildschirmspeichers

In dieser Speicherzelle steht eine Zahl, die als High-Byte dem Betriebssystem angibt, ab welcher Speicherzelle der Bildschirmspeicher beginnt.

Nach einem Kaltstart (nach dem Einschalten oder nach dem Drücken der RESET-Taste) steht hier eine 4, das ergibt als Anfangsadresse 1024 (= 4*256). Beim VC 20 ohne Erweiterung steht dort eine 30. Daraus folgt, daß die Anfangsadresse bei 7680 (= 30*256) liegt.

Der Bildschirmspeicher hat keinen absolut festen Platz. Innerhalb gewisser Grenzen kann er durch Verändern des Inhalts der Speicherzelle 53272 (36869 beim VC 20) verschoben werden. Die Methode dazu ist im Texteinschub näher beschrieben. Wichtig dabei ist, daß nach dem Verschieben der Inhalt der Speicherzelle 648 entsprechend geändert wird, damit auch das Betriebssystem die Verschiebung berücksichtigt.

Umgekehrt kann aber dem Betriebssystem durch Ändern der Zahl in der Speicherzelle 648 mitgeteilt werden, daß es Zeichen in einen Speicherbereich bringen soll, der außerhalb des »offiziellen«, durch die Speicherzelle 53272 (36869) festgelegten Bildschirmspeichers liegt.

Zwei Beispiele sollen das verdeutlichen. Der PRINT-Befehl macht letztlich nichts anderes, als viele Zahlen in den Bildschirm- und den Farbspeicher zu POKEn. Wenn nun der Zeiger in Zelle 648 verschoben wird, kann man mit einem PRINT-Befehl eine beliebige Zeichenkette außerhalb des Bildschirmspeichers speichern. Auf die gleiche Weise kann man beim C 64 Sprites mit einem PRINT-Befehl speichern, ohne mit READ viele lästige DATA-Zeilen lesen zu müssen.

Adresse 649 ($289)

Maximale Länge des Tastaturpuffers

Der Tastaturpuffer belegt, wie schon besprochen, die Speicherzellen 631 bis 640. Er kann darin maximal 10 Zeichen zwischenspeichern.

Der Inhalt der Speicherzelle 649 legt fest, wieviele Zellen des Tastaturpuffers verwendet werden sollen, eine Zahl also, die normalerweise zwischen 0 und 10 liegen sollte. Die 10 ist übrigens der Wert, welcher nach dem Einschalten vom Betriebssystem in die Zelle 649 gebracht wird.

Diese Zahl wird immer mit dem Inhalt der Speicherzelle 198 verglichen, der die aktuelle Anzahl der Zeichen im Tastaturpuffer angibt. Ist die Differenz der beiden Zahlen gleich Null, dann können keine weiteren Zeichen eingegeben werden.

Es ist naheliegend, daß durch Verändern der Zahl in Zelle 649 die Länge des Tastaturpuffers verändert werden kann. Der eine Extremfall ist 0:

POKE 649,0 schaltet die Tastatur aus. Nichts geht mehr.

Das kann bei Programmen oder Spielen, die durch falsches oder zeitlich unpassendes Drücken von Tasten gestört werden, recht nützlich sein. Einschalten kann man dann die Tastatur nur mit RUN/STOP und RESTORE.

Auch eine Erhöhung der Zahl in 649 über 10 hinaus ist möglich. Die Zeichen werden halt nur über die dafür reservierten Speicherzellen 631 bis 640 hinaus in Zellen geschrieben, die eigentlich eine andere Funktion haben. Bis zur Speicherzelle 645 geht das normalerweise ohne Probleme, da die betroffenen »fremden« Adressen nur direkt nach dem Einschalten des Computers gebraucht werden.

Probieren Sie es aus, indem Sie zuerst eine Zeitschleife laufen lassen und in dieser Zeit etwa 20 Tasten drücken. Am Ende der Zeitschleife wird der Inhalt des Tastaturpuffers ausgedruckt, und Sie sehen in der Tat 15 der eingegebenen Zeichen:

```
POKE 649,15
FOR X=0 TO 10000:NEXT X
QWERTYUIOPASDFGHJKL
```

Auf dem Bildschirm erscheinen die Zeichen Q bis G.

Wenn Sie die Zahl in 649 noch weiter erhöhen, dringen Sie in die Zellen 646 und 647 ein und diese bestimmen bekanntlich die Zeichenfarbe. Wenn Sie aber eine unbeabsichtigte und unkontrollierbare Farbänderung nicht stört, können Sie den Tastaturpuffer auf 17 Zeichen vergrößern. Ab 18 Zeichen stürzt der Computer ab.

Adresse 650 ($28A)

Flagge für Tastenwiederholung

Normalerweise steht in dieser Speicherzelle eine 0. Das bedeutet, daß die Funktion der Cursor-Tasten, der Leertaste und der INST/DEL-Taste wiederholt wird, solange die entsprechende Taste gedrückt wird.

Durch Verändern der Zahl in der Speicherzelle 650 kann diese Wiederholfunktion sowohl auf alle Tasten ausgedehnt als auch für alle Tasten gesperrt werden.

```
POKE 650,0
```

ist der Normalzustand, Wiederholfunktion für Cursor-, Leer- und INST/DEL-Taste

```
POKE 650,64
```

schaltet Wiederholfunktion für alle Tasten aus.

```
POKE 650,128
```

erweitert Wiederholfunktion auf alle Tasten.

Adresse 651 ($28B)

Zähler für Wiederholgeschwindigkeit der Tasten

Das Betriebssystem verwendet diese Speicherzelle als Zähler, der die Geschwindigkeit bestimmt, mit der eine Taste wiederholt wird, wenn sie länger gedrückt wird. Voraussetzung ist die durch Zelle 650 festgelegte Wiederholbarkeit der Taste.

Am Anfang steht in der Zelle 651 die Zahl 6. Sobald eine wiederholbare Taste gedrückt wird, zählt das Betriebssystem diese Zahl alle 0,0167 Sekunden (60mal in der Sekunde) um 1 zurück, bis die Zahl 1 erreicht ist. Dann erst wird das Zeichen der gedrückten Taste wieder auf den Bildschirm gedruckt oder ihre Funktion wiederholt.

Bei jedem folgenden Lauf steht in Zelle 651 die Zahl 4. Entsprechend verkürzt sich der Zählvorgang.

Am schnellsten würde die Wiederholung natürlich mit dem Wert 1 in der Speicherzelle 651 sein. Von Basic aus mit POKE 651,1 geht das leider nicht.

Im Texteinschub Nr. 27 »Turbo-Tasten« wird ein Maschinenprogramm beschrieben, welches dies kann.

Adresse 652 ($28C)

Zähler für die Ansprechzeit der Wiederholfunktion von Tasten

Diese Speicherzelle wird vom Betriebssystem als Zähler verwendet, der festlegt, wie lange eine wiederholbare Taste gedrückt sein muß, bis die Wiederholfunktion einsetzt.

Am Anfang steht in der Zelle 652 die Zahl 16. Diese Zahl wird alle 0,0167 Sekunden um 1 reduziert, bis die Zahl 0 erreicht ist. Dann wird das Zeichen der Taste auf den Bildschirm gebracht oder ihre Funktion wiederholt. Anschließend wird die Zahl 4 in die Speicherzelle 651 geschrieben (siehe dort), während die Zelle 652 so lange auf 0 stehen bleibt, bis eine andere Taste gedrückt wird. Wie diese anfängliche Verzögerung reduziert werden kann, steht im Texteinschub Nr. 27 »Turbo-Tasten«.

Adresse 653 ($28D)

Tastencode der SHIFT-, CTRL- und Commodore-Taste

In der Speicherzelle 203 stehen die Codes aller Tasten, die gedrückt werden, außer die der drei Steuertasten SHIFT, CTRL und Commodore (oft auch CBM-, Logo- oder C=-Taste genannt). Diese drei Ausnahmen haben ihr eigenes Code-Register, eben 653.

Der Grund dafür liegt in der Bedeutung der drei Tasten. Sie können ja bekanntlich verschiedene Zeichensätze einschalten:

- SHIFT schaltet das Zeichen vorne rechts auf einer Taste ein
- C= schaltet das Zeichen vorne links auf einer Taste ein
- CTRL schaltet die Farben vorn auf den Zahlentasten ein
- SHIFT + C= schaltet von dem normalen Zeichensatz auf die Groß-/Kleinschreibung um.

Ich habe diese Zusammenhänge auch bei der Behandlung der Speicherzellen 245 und 246 erwähnt.

Die Codezahlen selbst sind auch in der Tabelle 9 enthalten. Der Vollständigkeit halber sind sie hier noch einmal angegeben:

SHIFT	1
C=	2
CTRL	4
SHIFT und C=	3
SHIFT und CTRL	5
C= und CTRL	6
SHIFT und C= und CTRL	7

Mit dem folgenden kleinen Programm und mit ein wenig Fingerfertigkeit können Sie diese Codewerte nachvollziehen:

```
10 PRINT PEEK(653)
20 GOTO 10
```

Eine interessante Anwendung habe ich im Texteinschub Nr. 21 »Abfrage der Tastencodes oder 476 Funktionstasten« gegeben.

Adresse 654 ($28E)

Tastencode der zuletzt gedrückten SHIFT-, CTRL- oder C=-Taste

Diese Speicherzelle wird zusammen mit der Zelle 653 verwendet, um zu verhindern, daß ein schlechter Tastendruck als mehrfaches Drücken derselben Taste gedeutet wird. Im Fachdeutsch nennt man das »Entprellen« einer Taste oder eines Kontaktes. Die Funktion ist vergleichbar mit der der Zelle 197 gegenüber der Zelle 203 für alle anderen Tasten.

Adresse 655 und 656 ($28F und $290)

Vektor auf die Routine der Tastencode-Tabellen

Das Betriebssystem hat eine Routine ab Adresse 60232 (60380 beim VC 20), auf die der Vektor in 655 und 656 zeigt. Sie liest den Codewert der SHIFT-, CTRL- und C=-Taste in der Speicherzelle 653 aus und verändert entsprechend den Vektor der Zellen 245 und 246, so daß er auf die richtige Codetabelle zeigt.

Texteinschub Nr. 26
Bunte Zeichen und bunter Hintergrund

1) Bunte Zeichen

Wie Zeichen und Buchstaben in bunten Farben auf den Bildschirm gedruckt werden, lernt jeder Hobby-Programmierer schon bei den ersten Gehversuchen – dasselbe innerhalb eines Programms zu erreichen, dauert sicher schon etwas länger.

Bei der Diskussion der Speicherzelle 646 habe ich drei Methoden dafür erwähnt. Ich habe auch gesagt, daß ich die Methode, den Farbcodewert in die Speicherzelle 646 zu POKEn, für die eleganteste halte. Deswegen verwendet das folgende Demonstrations-Programm dieses Verfahren, um den Bildschirm mit einer bunten Reihe der Zahl 1 zu füllen.

```
10 PRINT CHR$(147)
20 POKE 53281,1
30 FOR J=0 TO 1000
40 POKE 646,INT(RND(1)*14+2)
50 PRINT "1";
60 NEXT J
```

VC 20-Besitzer müssen die Zeilen 20, 30 und 40 umändern in:

```
20 POKE 36879,233
30 FOR J=0 TO 505
40 POKE 646,INT(RND(1)*6+2)
```

Erklärung:

Zeile 10 löscht den Bildschirm, Zeile 20 erzeugt einen weißen Hintergrund und eine hellblaue Umrahmung. Zeile 30 zählt vom ersten bis zum letzten Platz auf dem Bildschirm. Zeile 40 erzeugt für jedes Zeichen auf dem Bildschirm eine neue Farbe. Zeile 50 schließlich druckt, durch das Semikolon gesteuert, die Zahl 1 hintereinander, und zwar in den Farben, die in Zeile 40 zufällig ausgewürfelt wurden.

RND(1)*14 erzeugt eine Zufallszahl zwischen 0,1 und 13,99. Der Befehl INT davor macht daraus eine ganze Zahl zwischen 0 und 13. Um aber die Codezahl 1 für Weiß zu vermeiden, addieren wir noch 2 dazu, so daß wir Farbcodes zwischen 2 und 15 erhalten. Beim VC 20 ist das alles auf die Farben 2 bis 7 beschränkt.

Das Ergebnis ist wie gesagt ein Bildschirm voller Einser, deren Farben bunt wie ein Regenbogen abwechseln.

2) Bunter Hintergrund

Bunte Zeichen stellen also kein Problem dar. Wie steht es aber mit einem bunten Hintergrund? Den können wir zwar auch verändern (POKEn der Speicherzelle 53281 beziehungsweise 36879 beim VC 20), aber es bleibt immer nur »eintönig«.

Vom Commodore-Autor Jim Butterfield kenne ich nun eine Methode, die auch einen vielfarbigen Hintergrund bietet.

Butterfield geht dabei von einer lustigen Überlegung aus. Wir wissen zum Beispiel, daß der nächtliche Sternenhimmel aus hellen Punkten besteht, die vor einem schwarzen Hintergrund leuchten. Ohne dieses Wissen könnten wir aber ebenso gut annehmen, daß der Himmel – also der Hintergrund – im hellsten Weiß erstrahlt, aber durch einen schwarzen Vorhang (Vordergrund) mit vielen kleinen Löchern abgedunkelt ist.

Das folgende Demo-Programm benutzt diese Denkweise.

```
100 PRINT CHR$(147)
110 POKE 53281,1
120 FOR J=0 TO 1000
130 POKE 1024+J,160
140 POKE 55296+J,INT(RND(1)*14+2)
150 NEXT J
160 FOR K=0 TO 1000
170 POKE 1024+K,177
180 NEXT K
```

Für den VC 20 (ohne Erweiterung) sieht das Programm so aus:

```
100 PRINT CHR$(147)
110 POKE 36879,233
120 FOR J=0 to 505
130 POKE 7680+J,160
140 POKE 38400+J,INT(RND(1)*6+2)
150 NEXT J
160 FOR K=0 TO 505
```

```
170 POKE 7680+K,177
180 NEXT K
```

Die ersten drei Zeilen sind mit denen des ersten Demonstrations-Programms identisch.

Zeile 130 und 140 setzen auf jeden Platz des Bildschirms zuerst ein invertiertes Leerzeichen (Bildschirmcode 160), und zwar in einer der vielen möglichen Farben, per Zufallsgenerator in Zeile 140 ausgewählt.

Leerzeichen mit Farbe? Zugegeben, ein Leerzeichen hat normalerweise keine Farbe, man sieht es nicht. Das invertierte Leerzeichen hat aber eine Farbe. Sie kennen es vom Cursor, dessen Blinken dadurch erzeugt wird, daß das Leerzeichen zwischen normal und invertiert umgeschaltet wird (siehe auch die Beschreibung der Speicherzelle 647). Auf diese Weise besteht jetzt der Bildschirm aus einer Vielzahl von bunten Quadraten. Das ist der Vorhang von Jim Butterfield, der vor dem hellen weißen Hintergrund hängt.

Ab Zeile 160 werden alle Plätze des Bildschirms mit der invertierten 1 (Bildschirmcode 177) gefüllt. Diese invertierten Zeichen sind in der Farbe des Hintergrundes geschrieben, eben weiß. Dadurch entsteht der Eindruck, als wäre der Hintergrund bunt und die Zeichenfarbe weiß.

Der Eindruck verstärkt sich noch, wenn wir die 1 über den Bildschirm wandern lassen. Das erreichen wir durch Ändern der folgenden Zeilen:

```
170 POKE 1024+K,160
175 POKE 1025+K,177
```

Durch geschicktes Ausbauen der Zeile 140 können Sie einen vielfarbigen Bildschirm-Hintergrund in Zeilen oder Blöcken erzielen, ein weites Gebiet für bunte Grafik.

Texteinschub Nr. 27
Turbo-Tasten

Das Trio der Speicherzellen 650, 651 und 652 ist zuständig für die Steuerung der sogenannten Wiederholfunktion der Tasten. Darunter verstehen wir die Eigenschaft der Tastatur, das Zeichen oder die Funktion einer Taste so lange zu wiederholen, bis die Taste losgelassen wird. Normalerweise haben diese Funktion nur die Leertaste, die Cursor-Tasten und die INST/DEL-Taste.

Die Zahl in Speicherzelle 650 entscheidet, welche Tasten wiederholbar sind.

Schalten Sie bitte mit POKE 650,128 alle Tasten auf »wiederholbar« um.

Wenn Sie jetzt eine Taste drücken und sie festhalten, werden Sie folgendes beobachten können.

Nachdem das erste Zeichen auf dem Bildschirm erschienen ist, vergeht eine kurze Zeit, erst dann wird es mit einer gleichbleibenden Geschwindigkeit immer wieder ausgedruckt.

Für die anfängliche Verzögerung ist die Speicherzelle 652, für die Geschwindigkeit der nachfolgenden Wiederholungen die Speicherzelle 651 zuständig.

Viele Spieler und Anwender haben sich sicher schon oft gewünscht, sowohl die Reaktionszeit als auch die Geschwindigkeit der Wiederholfunktion beschleunigen zu können. Leider geht es in Basic nicht, weil die Zahlen in den Zellen 651 und 652 60mal in der Sekunde auf ihren ursprünglichen Wert zurückgesetzt werden.

Aber in Maschinensprache geht es sehr wohl, und zwar mit der sogenannten Interrupt-Methode. Über sie und ihre Wirkungsweise ist schon ausführlich berichtet worden: von Boris Schneider in Ausgabe 3/85 (Der gläserne VC 20) und von Heimo Ponnath in den Ausgaben 7/85 und 8/85 (Assemblerkurs). Ich werde hier nur innerhalb der Beschreibung des folgenden Kochrezeptes darauf eingehen.

Das Kochrezept zur Veränderung der Inhalte von 651 und 652 stammt von Dan Carmichael aus seinem Aufsatz »Speeding Up The VIC« in Ausgabe 10/83 der COMPUTE!'s Gazette.

Wir schreiben es als Maschinenprogramm in Form von DATA-Zeilen in den Bandpuffer ab Adresse 828, wo es geschützt residieren kann, solange keine Kassettenoperationen durchgeführt werden. Das Ladeprogramm in Basic steht in Listing 1 (Seite 63). Für den VC 20 lautet die vorletzte Zahl 191 statt 49.

In Listing 2 (Seite 63) ist das Programm disassembliert dargestellt.

Beim VC 20 lautet der Sprungbefehl in Zelle 851 JMP 60095.

Für Anhänger der hexadezimalen Darstellung gebe ich das Programm als HEX-Ausdruck in Listing 3 (Seite 63) wieder.

Für den VC 20 lautet die letzte Zeile anders:

,0353 4C BF EA JMP EABF

Mit dem Befehl SEI werden jegliche Programmunterbrechungen gesperrt. Anschließend kommt das Zahlenpaar 73 und 3 in die Speicherzellen 788 und 789, wo es in Low/High-Byte Darstellung die Adresse 841 (73+256*3=841) darstellt.

In 788 und 789 steht normalerweise ein Vektor auf die Adresse 59953 (60095 beim VC 20), von der aus die Aufgaben der »normalen« Unterbrechungsroutine gesteuert werden. Wir »verbiegen« also den Vektor so, daß er auf die Speicherzelle 841 zeigt.

Die schon genannte Unterbrechungsroutine, die 60mal pro Sekunde alles unterbricht, um die STOP-Taste abzufragen, die Uhr weiterzuschalten und so weiter, springt jetzt nicht auf 59953, sondern zuerst nach 841.

Ab 841 steht jedoch der zweite Teil unseres Maschinenprogramms, das die eingangs gewünschte 1 beziehungsweise 0 nach 651 und 652 schreibt. Das erfolgt jetzt laufend, ein Effekt, der uns in Basic verwehrt ist. Danach allerdings kommt ein letzter Sprungbefehl, der dort weitermacht, wo die Unterbrechunsroutine ursprünglich hätte fortfahren sollen, nämlich in 59953 (60095).

Jetzt fehlt nur noch die Beschreibung, wie sich das alles auswirkt. Ich nehme an, Sie haben immer noch mit POKE 650,128 die gesamte Tastatur auf Wiederholfunktion geschaltet, wenn nicht, holen Sie es bitte nach. Laden Sie das Basic-Programm von Listing 1 und starten Sie es mit RUN. Jetzt steht es in den Speicherzellen 828 bis 853 und kann mit SYS 828 gestartet werden.

Wenn Sie jetzt wieder eine Taste länger gedrückt halten, flitzt das entsprechende Zeichen wie ein Turbo-Auto über den Bildschirm. Der Cursor ist mit den Augen fast nicht mehr zu verfolgen. Es geht alles so schnell, daß Sie Mühe haben, nur ein einzelnes Zeichen auf den Bildschirm zu bringen. Wenn Sie das wollen: Mit RUN/STOP und RESTORE stellen Sie den ursprünglichen Zustand wieder her.

Das kleine Maschinenprogramm läßt sich in jedes Spiel oder Anwendungsprogramm nutzbringend einbauen.

Es gibt Anwenderprogramme, die diesen Vektor so verbiegen, daß die Decodierung der Tasten umgangen und durch eine andere, selbstgebaute Routine ersetzt wird. So kann zum Beispiel das Drücken einer bestimmten Taste umgemünzt werden in Ausdrucken von Basic-Befehlen auf dem Bildschirm.

Adresse 657 ($291)

Flagge für Verriegelung der Zeichensatz-Umschaltung

Durch gleichzeitiges Drücken der SHIFT- und der Commodore-Taste wird bekanntlich der Zeichensatz 1 (Großbuchstaben und Grafik-Zeichen) umgeschaltet auf den Zeichensatz 2 (Groß- und Kleinbuchstaben), ein zweites Drücken der beiden Tasten schaltet den Zeichensatz zurück.

Diese Umschaltung wird verriegelt, wenn in der Speicherzelle 657 eine 128 steht. Eine 0 läßt die Umschaltung zu.

Dieser Effekt kann auf zwei, beim C 64 sogar auf drei Arten erzielt werden:

☐ Umschaltung des Zeichensatzes zulassen

```
POKE 657,0
PRINT CHR$(9)
CTRL und I (nur C 64)
```

☐ Umschaltung des Zeichensatzes verriegeln

```
POKE 657,128
PRINT CHR$(8)
CTRL und H (nur C 64)
```

Adresse 658 ($292)

Flagge für Scrollen

Die Flagge in dieser Speicherzelle legt fest, ob eine weitere echte Zeile zu einer logischen Zeile hinzugefügt wird, sobald der Cursor über das 40ste Zeichen der Zeile (22ste Zeichen beim VC 20) hinausläuft.

Steht in 658 eine 0, dann werden alle Zeilen hochgeschoben (man nennt das »scrollen«), um der neuen Zeile Platz zu machen.

Wenn in der Zeile irgendein Wert größer als Null steht, unterbleibt dieses Scrollen. Die Flagge wird immer dann auf den höheren Wert gesetzt, wenn Zeichen im Tastaturpuffer (631 bis 640) stehen und darauf warten, am Ende des Programms ausgedruckt beziehungsweise ausgeführt zu werden. Diese Verriegelung wird deshalb eingesetzt, weil im Tastaturpuffer Zeichen wie zum Beispiel Cursor-Bewegungen stehen können.

Von Basic aus kann diese Speicherzelle nicht beeinflußt werden.

Die RS232-Schnittstelle

Wir haben schon früher Speicherzellen besprochen, die mit der RS232-Schnittstelle zu tun haben, ohne die letztere dabei genauer zu betrachten. Da der heutige Teil des Kurses jedoch die zehn wichtigsten Speicherzellen behandelt, welche diese Schnittstelle betreffen, komme ich nicht umhin, näher auf sie einzugehen. Ich tue das mit voller Absicht, obwohl dadurch die erklärenden Texteinschübe mehr Platz einnehmen als die Besprechung der Speicherstellen selbst. Falls Sie mit Schnittstellen des Computers nicht vertraut sind, sollten Sie zuerst den Texteinschub Nr. 28 »Schnittstelle und Port« anschauen, mit dem ich die gängige Begriffsverwirrung klären möchte.

Adresse 659 ($293)

RS232-Steuerregister

Jeder OPEN-Befehl, mit dem bekanntlich eine Datei (File) eröffnet wird, kann neben File-Nummer und Geräte-Nummer auch einen File-Namen haben. Der File-Namen einer RS232-Schnittstelle hat maximal nur vier Zeichen. Das erste Zeichen wird in diese Speicherzelle 659 gebracht und steuert dort Übertragungsgeschwindigkeit, die Wortlänge und die Anzahl der Stopp-Bits. Die nähere Bedeutung dieser Fachwörter können Sie dem Texteinschub Nr. 29 »Die Elemente der RS232-Schnittstelle« entnehmen. Tabelle 11 zeigt die Bedeutung jedes einzelnen Bits dieser Speicherzelle.

Die praktische Anwendung dieser Bit-Werte innerhalb eines OPEN-Befehls ist ausführlich im Texteinschub Nr. 30 »Die Programmierung der RS232-Schnittstelle« beschrieben.

Adresse 660 ($294)

RS232-Befehlsregister

In diese Speicherzelle wird, ähnlich wie bei der Zelle 659, das zweite Zeichen des File-Namens gebracht. Die einzelnen Bits steuern das Handshake-Protokoll (3-/X-Leitung), den Duplex-Modus (Halb-/Voll-Duplex) und die Parity-Prüfung (keine, gerade, ungerade). Die nähere Bedeutung dieser Fachwörter können Sie dem Texteinschub Nr. 11 »Die Elemente der RS232-Schnittstelle« entnehmen. Tabelle 12 zeigt die Bedeutung jedes einzelnen Bits dieser Speicherzelle.

Wenn Sie sich für die praktische Anwendung dieser Bit-Werte innerhalb eines OPEN-Befehls interessieren, dann verweise ich an dieser Stelle auf den Texteinschub Nr. 30 »Die Programmierung der RS232-Schnittstelle«.

Adresse 661 und 662 ($295 und $296)

RS232 frei wählbare Übertragungsgeschwindigkeit

Es war ursprünglich vorgesehen, durch entsprechende Wahl des dritten und vierten Zeichens im File-Namen beliebige Übertragungsgeschwindigkeiten einzustellen. Die jeweiligen Werte sollten die Speicherzellen 661 und 662 enthalten. Diese Möglichkeit wurde aber nicht eingebaut. Der Grund dafür dürfte wohl der sein, daß die wählbaren Übertragungsgeschwindigkeiten aller Geräte auf bestimmte Werte normiert sind.

Adresse 663 ($297)

RS232-Statusregister

Genauso wie in der Speicherzelle 144 der Status aller Ein- und Ausgabe-Operationen angezeigt wird, werden alle Fehler der RS232-Schnittstelle in der Speicherzelle 663 angezeigt. Die Bedeutung der einzelnen Bits, wenn sie auf 1 gesetzt sind, zeigt Tabelle 14.

Der Status wird nicht automatisch angezeigt, sondern muß vom Programm abgefragt werden. Abfragen können Sie sowohl durch PEEKen der Speicherzelle 663 als auch durch Aufrufen der Statusvariablen ST. Die Variable ST, die normalerweise den Inhalt der Zelle 144 wiedergibt, schaltet nach dem Eröffnen eines RS232-Kanals durch OPEN 1,2 auf die Speicherzelle 663 um. Jedoch ist Vorsicht geboten, da durch Aufruf von ST der Inhalt von 663 gelöscht wird.

Es ist ratsam, den Wert von ST erst einer anderen Variablen zuzuordnen, wenn sie mehrfach verwendet werden soll.

Falls das Status-Register einen Fehler anzeigt, muß das Programm entsprechende Konsequenzen ziehen. Wenn zum Beispiel Bit 0 oder Bit 1 gesetzt sind, ist es angebracht, das letzte Daten-Byte noch einmal zu übertragen. Wenn Bit 2 gesetzt ist, heißt dies, daß der GET#-Befehl den Eingabepufferspeicher nicht schnell genug entleert. Falls die Übertragungsgeschwindigkeit von 300 Bit/s, die maximal mit einem Basic-Programm erreichbar ist, nicht ausreicht, muß entweder der Sender langsamer eingestellt werden, oder Sie schrei-

BINÄR	BITWERT	STEUERFUNKTION
Bit 0 bis 3 steuern die Übertragungsgeschwindigkeit		
xxxx0000	0	(siehe 1)
xxxx0001	1	50 Bit/s
xxxx0010	2	75 Bit/s
xxxx0011	3	110 Bit/s
xxxx0100	4	134,5 Bit/s
xxxx0101	5	150 Bit/s
xxxx0110	6	300 Bit/s
xxxx0111	7	600 Bit/s
xxxx1000	8	1200 Bit/s
xxxx1001	9	1800 Bit/s
xxxx1010	10	2400 Bit/s
die Werte 11 bis 15 sind nicht belegt		
Bit 4 ist nicht belegt		
Bit 5 und 6 steuern die Wortlänge		
x00xxxxx	0	8-Bit-Wort
x01xxxxx	32	7-Bit-Wort
x10xxxxx	64	6-Bit-Wort
x11xxxxx	96	5-Bit-Wort
Bit 7 steuert die Stopp-Bits		
0xxxxxxx	0	1 Stopp-Bit
1xxxxxxx	128	2 Stopp-Bit

Fußnote (1)
Bei diesem Bitwert sollte ursprünglich der Anwender in Speicherzelle 661 eigene Bandraten wählen können. Diese Möglichkeit ist aber nicht realisiert worden.

Tabelle 11. Die Bedeutung der einzelnen Bits im RS232-Steuerregister

BINÄR	BITWERT	STEUERFUNKTION
Bit 0 steuert den Handshake-Typ		
xxxxxxx0	0	3-Leitung
xxxxxxx1	1	X-Leitung
Bit 1 bis 3 sind nicht belegt		
Bit 4 steuert den Duplex-Modus		
xxx0xxxx	0	Voll-Duplex
xxx1xxxx	16	Halb-Duplex
Bit 5 bis 7 steuern den Paritäts-Test		
000xxxxx	0	keine Parität
001xxxxx	32	ungerade (1)
010xxxxx	64	keine Parität
011xxxxx	96	gerade (1)
100xxxxx	128	keine Parität
101xxxxx	160	Parität=1 (2)
110xxxxx	192	keine Parität
111xxxxx	224	Parität=0 (2)

Fußnoten:
(1) vom Sender berechnet, vom Empfänger geprüft
(2) vom Sender festgelegt, vom Empfänger nicht geprüft

Tabelle 12. Das RS232-Befehlsregister im einzelnen

ben das Programm in Maschinensprache.

Adresse 664 ($298)

RS232 - Anzahl der zu übertragenden Bits

Diese Speicherzelle wird verwendet, um festzustellen, mit wievielen Nullen das zu übertragende Zeichen aufgefüllt werden muß, um die in Speicherzelle 659 (Bit 5 und 6) ausgewählte Wortlänge herzustellen (s. Speicherzellen 168 und 180).

BIT	BIT-WERT	BEDEUTUNG
0	1	Fehler bei Paritäts-Test
1	2	Fehler in der Bit-Folge
2	4	Überlauf des Eingabepufferspeichers
3	8	Eingabepufferspeicher ist leer
4	16	das Clear-To-Send (CTS)-Signal (Handshake) fehlt
5	32	nicht belegt
6	64	das Data-Set-Ready (DSR)-Signal (Handshake) fehlt
7	128	Übertragung ist unterbrochen

Tabelle 14. Das RS232-Statusregister

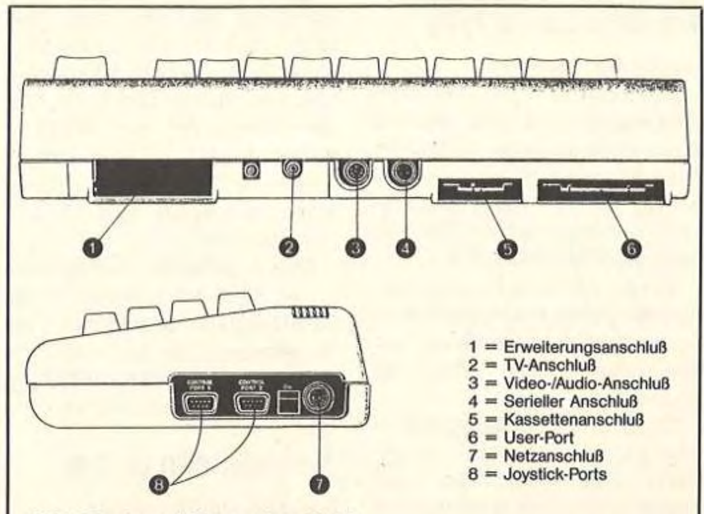


Bild 23. Anschlüsse des C 64

Texteinschub Nr. 28
Schnittstelle und Port

Immer wenn die Rede davon ist, den Computer mit irgendwelchen Geräten zu verbinden, tauchen Fachwörter auf, wie Interface, Schnittstelle, Port, Eingang, Ausgang und Stecker. Da im Kurs gerade die Speicherzellen behandelt werden, die für die RS232-Schnittstelle zuständig sind, möchte ich die Gelegenheit nutzen, ein wenig Klarheit in dieses Begriffswirrwarr zu bringen. Zuerst sollen die Begriffe erklärt werden:

1) »Interface« ist das englische Wort für »Schnittstelle«.

In einer Schnittstelle sind die Regeln und Vorschriften festgelegt, wie Daten zwischen zwei Geräten (zum Beispiel Computer und Floppy) ausgetauscht werden. Festgelegt ist hauptsächlich:
- ob ein Datenwort auf einen Schlag (parallel) oder jedes Bit einzeln (seriell) übertragen wird
- die Geschwindigkeit der Übertragung
- die Signale, mit denen die beteiligten Geräte den Ablauf der Übertragung steuern
- mit welchen Spannungs- oder Stromwerten die binäre 1 beziehungsweise 0 dargestellt wird
- die elektrischen Spannungen und Ströme, die bei der Übertragung maximal auftreten dürfen

Sie sehen, eine Schnittstelle ist in erster Linie eine Anzahl von Regeln. Manchmal allerdings werden auch die Module und Spezialkabel, welche die Regeln technisch in die Tat umsetzen, Schnittstellen genannt.

2) Über einen »Ausgang« kann der Computer (oder ein anderes Gerät) Daten abgeben, über einen »Eingang« erhält er Daten. Ein »Port« ist beides, Ein- und Ausgang. Ein »Stecker« schließlich ist die technische Ausführung der Verbindung.

So, nach dieser Begriffserklärung wollen wir uns anschauen, welche Ports, Ein- und Ausgänge, der Computer hat. Die Zeichnung dieser Anschlüsse (Bild 23) habe ich dem Commodore-Handbuch entnommen, nicht aber ihre Bezeichnungen, denn diese gehen bereits wild durcheinander.

Der »Erweiterungsanschluß« (1) wird hauptsächlich als Eingang für Spielmodule verwendet. Er ist aber ein echter Port, nicht zuletzt zur Speichererweiterung beim VC 20 und kann für Schnittstellen über entsprechende Routinen des Betriebssystems programmiert werden.

Der »TV-Anschluß« (2) ist ein reiner Ausgang des im Computer eingebauten Fernsehmodulators, der beim VC 20 fehlt.

Der »Video/Audio-Anschluß« (3) ist ebenfalls ein reiner Ausgang der Ton- und Bildsignale für einen Monitor oder für den externen Fernsehmodulator des VC 20.

Der »Serielle Anschluß« (4) ist ein Port, über den das Diskettenlaufwerk und Drucker angeschlossen werden. Er ist für Schnittstellen programmierbar.

Der »Kassettenanschluß« (5) ist ebenfalls ein Port, der speziell für die Datasette eingerichtet ist. Bastler und Tüftler, die Schaltpläne lesen können und das Betriebssystem des Computers kennen, müßten in der Lage sein, mit diesem Port auch andere externe Geräte zu steuern. Für die genormten Schnittstellen kommt er meines Wissens nicht in Frage.

Der »User-Port« (6) ist das, was sein Name sagt, nämlich ein Port für verschiedene Anwendungen und Schnittstellen. Er ist über 16 Register des 6526 Complex Interface-Adapters (CIA) mit den Adressen 56320 bis 563215 frei programmierbar. Der VC 20 hat dafür einen 6522 Versatile Interface-Adapter (VIA), dessen 16 Register die Adressen 37136 bis 37151 haben.

Der »Netzanschluß« (7) ist ein reiner Eingang, aber nicht für Daten.

Die »Spielanschlüsse« (8) (nur einer beim VC 20) werden eigentlich nur als Eingang für Joysticks, Lichtgriffel und Paddles (Drehregler) verwendet, obwohl sie vom Prinzip her programmierbare Ports sind. Ihre universelle Verwendung ist sicher nur Spezialisten vorbehalten.

Zuletzt sollen auch die verbreitetsten Schnittstellen noch erwähnt werden. International haben sich besonders die folgenden drei Schnittstellen durchgesetzt:
- die RS232-Schnittstelle
- die IEC/IEEE-488-Schnittstelle
- die Centronics-Schnittstelle

Die »RS232-Schnittstelle« ist eine serielle Schnittstelle. Sie wurde schon vor 100 Jahren als »TTY-Version« zur Textübertragung mit Fernschreibern eingerichtet, bei der die logische 0 durch einen Strom von 20 Milliampere und die 1 durch keinen Strom dargestellt wurde. Heute wird fast nur noch die »V.24-Version« zur Datenfernübertragung verwendet, bei der die 0 durch eine positive Spannung zwischen 3 und 15 Volt, die 1 aber durch eine entsprechende negative Spannung dargestellt wird.

Beim C 64 und VC 20 ist die RS232-/V.24-Schnittstelle am User-Port verfügbar, allerdings nicht mit den oben genannten Spannungswerten für die 0 und 1. Dieses für Commodore typische Sparverfahren macht eine zusätzliche Signalumsetzung erforderlich. Die RS232-Schnittstelle wird hauptsächlich für Datenübertragung per Modem oder Akustikkoppler eingesetzt. Ihr Arbeitsprinzip ist im Texteinschub Nr. 29 »Die Elemente der RS232-Schnittstelle« beschrieben.

Zur parallelen Datenübertragung entstand in Europa die »IEC-625-Schnittstelle«, in USA die »IEEE-488-Schnittstelle«. Beide sind praktisch identisch. Sie unterscheiden sich nur in der Verwendung des Steckers (was natürlich idiotisch ist). Bei den Commodore-Computern ist diese Schnittstelle sowohl am Erweiterungs-Port (1) als auch über den seriellen Port (4) einrichtbar. Der serielle Port allerdings enthält wiederum eine für Commodore typische Einschränkung. Er erlaubt, wie sein Name

andeutet, nur eine serielle Datenübertragung. Das heißt, statt – wie bei der IEC-/IEEE-Schnittstelle festgelegt – alle 8 Bit eines Wortes über acht Leitungen gleichzeitig, werden hier die Bits hintereinander auf nur einer Leitung übertragen. Auch das erfordert eine zusätzliche Anpassung. Das Prinzip der IEC-/IEEE-Schnittstelle wurde bereits in Ausgabe 3/85 ab Seite 24 von Arnd Wängler genau beschrieben.

Die »Centronics-Schnittstelle« ist aus der harten Realität des Geschäftslebens entstanden. Während sich noch die Normstellen in Europa und USA herumstritten, hat der damalige Marktführer unter den Druckerherstellern, die Firma Centronics, eine eigene Schnittstelle geschaffen, die sich schlicht und einfach durch die weite Verbreitung der Centronics-Drucker durchgesetzt hat. Sie ist eine parallele Schnittstelle, die sich beim C 64/VC 20 sowohl am User-Port (6) als auch am seriellen Port (4) einrichten läßt. Zur Beschreibung der Schnittstelle habe ich in meiner Literatursammlung nur zwei Aufsätze gefunden, einen von Georg Werner in c't, Ausgabe 4/84, Seite 92, einen anderen von Peter Bonsch in Computer Persönlich, Ausgabe 11/83 ab Seite 152. Tabelle 13 gibt eine Zusammenfassung über die Realisierbarkeit der drei Schnittstellen an den Ports von C 64 und VC 20.

SCHNITT-STELLE	PORT	Erweit. (1)	Seriell (4)	Kassett. (5)	User (6)	Spiel (8)
RS232					X	
IEC/IEEE		X	X			
Centronics			X		X	

Tabelle 13. Realisierbarkeit der Schnittstellen am C 64 und VC 20

Texteinschub Nr. 29
Die Elemente der RS232-Schnittstelle

Da meine Texteinschübe kurz sein sollen, beschränke ich mich auf Erklärungen der Vorgänge, die mit den im Kurs behandelten Speicherzellen 659 bis 670 zu tun haben. Weitere Erläuterungen können Sie dem Aufsatz von Jens Maßmann der Ausgabe 5/85, Seite 80 entnehmen.

Wortlänge: Die Schnittstelle ermöglicht die Übertragung von Datenwörtern (Bytes), deren Länge vor der Übertragung eingestellt werden kann. Es sind Wortlängen von 5, 6, 7 oder 8 Bit erlaubt. Die Wortlänge wird durch Bit 5 und 6 der Speicherzelle 659 eingestellt.

Übertragungsgeschwindigkeit (Baud-Rate): Daten werden seriell übertragen, das heißt alle Bits eines Datenwortes (Byte) laufen hintereinander über eine Leitung zum Empfänger. Dabei ist wesentlich, daß Sender und Empfänger sich einig sind, mit welcher Geschwindigkeit die Bit-Kette übertragen wird. Diese Übertragungsgeschwindigkeit, auch Baud-Rate genannt, wird in Bit/s angegeben. Die Übertragungsgeschwindigkeit wird durch Bit 0 bis 3 der Speicherzelle 659 eingestellt und reicht von 110 bis 2400 Bit/s.

Stopp-Bits: Die Übertragungsgeschwindigkeit muß sowohl im Sender als auch im Empfänger der Daten fest eingestellt werden. Da die beiden Geräte dies unabhängig voneinander tun, besteht die Gefahr, daß diese Einstellungen nicht ganz genau gleich sind. Das könnte zur Folge haben, daß sie nach vielleicht 2000 Bit um 1 Bit auseinanderliegen. Alle nachfolgenden Übertragungen wären dann völlig falsch.

Deshalb wird die Übertragung nach jedem Wort neu eingestellt, man nennt das »synchronisieren«. Dazu dient am Anfang eines Wortes ein Start-Bit und am Ende eines Wortes ein oder zwei Stopp-Bits. Die Stopp-Bits definieren den Ruhezustand (logische 1) der Übertragungsleitung, ihre Anzahl die minimale Zeit des Ruhezustandes. Sobald ein neues Wort mit einem Start-Bit (logische 0) beginnt, übernimmt der Empfänger den Übergang von 0 nach 1 als Startimpuls für die Abfrage der nächsten ankommenden Bits. Die Anzahl der Stopp-Bits werden durch Bit 7 der Speicherzelle 659 eingestellt.

Paritätsprüfung: Zusätzlich zu den Datenbits und den Start-/Stopp-Bits können sogenannte Paritätsbits übertragen werden. Sie ermöglichen eine grobe Fehlerkontrolle. Der Sender errechnet die Quersumme aller Datenbits. Bei der sogenannten »geraden« Paritätsprüfung wird das Paritätsbit so gewählt, daß es die Quersumme zu einer geraden Zahl ergänzt, bei der »ungeraden« Paritätsprüfung ist es gerade umgekehrt. Der Empfänger macht dieselbe Rechnung und vergleicht sein Paritätsresultat mit dem empfangenen Paritätsbit des Senders. Sie sollten natürlich gleich sein. Auf diese Weise können einfache Übertragungsfehler erkannt werden. In den Commodore-Computern sind noch zwei weitere Möglichkeiten eingebaut, nämlich das Paritätsbit ohne Quersummenrechnung immer auf 1 oder aber immer auf 0 zu setzen. Mit den Bits 5 bis 7 der Speicherzelle 660 können insgesamt vier verschiedene Paritätsprüfungen eingestellt werden (siehe auch Texteinschub Nr. 18).

Duplex-Modus: Sind zwei Geräte, von denen eines nur empfangen, nicht aber selbst senden kann, über eine Leitung verbunden, nennt man diese Einbahnstraße eine Simplex-Verbindung. Können aber beide Geräte senden und empfangen, spricht man von einer Duplex-Verbindung.

Duplex gibt es in zwei Betriebsarten. Der Voll-Duplex-Modus erlaubt ein gleichzeitiges Senden beider Geräte. Im Halb-Duplex-Modus kann immer nur ein Gerät senden, allerdings wechselweise. Der Duplex-Modus wird durch Bit 4 der Speicherzelle 660 eingestellt.

Handshake-Protokoll: Mit Handshake (Händeschütteln) wird ein Verfahren bezeichnet, bei dem zwei Geräte sich gegenseitig durch gesonderte Signale mitteilen, ob sie bereit sind, Daten abzusenden beziehungsweise zu empfangen. Die Festlegung der zeitlichen Reihenfolge dieser Handshake-Signale nennt man Protokoll.

Dieses Verfahren hat den Vorteil, daß Sender und Empfänger völlig unabhängig voneinander ihr eigenes Programm ausführen können und nur selten aufeinander warten müssen. Voraussetzung ist allerdings ein Pufferspeicher (siehe unten). Es gibt zwei Arten von Handshakes, den 3-Leitungs-Handshake (auch Rückkanal-Handshake genannt) und den X-Leitungs- oder Voll-Handshake.

Der 3-Leitungs-Handshake braucht, wie der Name sagt, nur drei Leitungen: für gemeinsame Erde (Masse), für die gesendeten Daten und für die empfangenen Daten. Der Handshake besteht darin, daß der Empfänger auf der freien Leitung, eben dem Rückkanal, dem Sender durch je ein Zeichen mitteilt, wenn er bereit ist, Daten zu übernehmen oder wenn er keine Daten übernehmen kann.

Der X-Leitungs-Handshake stellt viel mehr Leitungen zur Verfügung, nämlich die gleichen drei wie vorher, zusätzlich aber pro Sender und Empfänger mehrere Leitungen für Anmeldung und Rückmeldung der Bereitschaft, sowie für die Ausführung der Übertragung. Es gibt theoretisch insgesamt 25 Leitungen für die RS232-Schnittstelle, beim C 64 beziehungsweise VC 20 sind aber nur zehn ausgeführt. Das Handshake-Protokoll kann durch Bit 0 der Speicherzelle 660 ausgewählt werden.

Pufferspeicher: Immer wenn ein RS232-Kanal geöffnet wird, zwackt das Betriebssystem des Computers dem Programmspeicher am oberen Ende zwei Pufferspeicher ab, mit einer Größe von je 256 Byte für empfangene und zu sendende Daten. Diese First-In-First-Out-Speicher (die als erste eingespeicherten Daten werden auch als erste wieder ausgelesen) sind als dynamische Ringspeicher aufgebaut. Statt zu warten, bis der Empfänger zur Datenübernahme bereit ist, schreibt der Sender die Daten in den Pufferspeicher, aus dem die Schnittstelle sie an den Empfänger weitergibt, sobald dieser bereit ist.

Dieses fast ungeordnete Füllen und Leeren des Pufferspeichers hat zur Folge, daß Beginn und Ende des Speichers je nach Datenmenge innerhalb der 256 Byte stets in Bewegung sind. Um jederzeit die Anfangs- und Endadressen feststellen zu können, werden sie in den Speicherzellen 667 bis 670 mitgezählt.

Statusregister: In der Speicherzelle 663 werden alle Fehler einer RS232-Übertragung festgehalten. Jedes Bit hat eine eigene Bedeutung, die in der Tabelle bei der Beschreibung der Speicherzelle 663 angegeben ist. Diese Fehler werden leider nicht, wie im Basic, automatisch angezeigt. Sie müssen vielmehr ausgelesen und identifiziert werden, um dann im Programm mit entsprechenden Maßnahmen korrigiert zu werden.

Texteinschub Nr. 30
Die Programmierung der RS232-Schnittstelle

Die Programmierung der RS232-Schnittstelle ist denkbar einfach. Alle dazu notwendigen Routinen sind im Betriebssystem des Computers bereits enthalten. Das genau macht ja die Schnittstelle so attraktiv. Die Schnittstelle verwendet genau dieselben Befehle wie die serielle Schnittstelle, über die der Computer mit Floppy und Drucker verbunden ist, nämlich OPEN, CMD, PRINT#, GET#, INPUT# und CLOSE. Auch die Statusvariable ST wird herangezogen. Wichtig ist jedoch, daß die RS232-Schnittstelle die Gerätenummer 2 hat.

Eröffnung des RS232-Kanals

Wie gewohnt, wird er mit dem OPEN-Befehl geöffnet:

OPEN File-Nr, Geräte-Nr, Datenkanal-Nr, File-Name

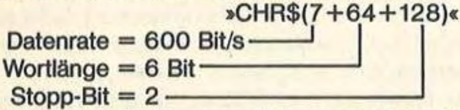
- die File-Nummer kann Werte von 0 bis 255 annehmen, wie bei jedem OPEN-Befehl auch
- die Geräte-Nummer ist immer 2
- der Wert der Datenkanal-Nummer ist bedeutungslos, da immer nur ein RS232-Kanal offen sein darf. Wird zusätzlich ein zweiter Kanal geöffnet, werden die Daten des ersten Kanals im Pufferspeicher zerstört.
- der File-Name hat hier eine besondere und entscheidende Bedeutung. Er besteht aus maximal vier Zeichen. Der ASCII-Wert des ersten Zeichens wird in die Speicherzelle 659 übertragen und legt dadurch die Übertragungsgeschwindigkeit, die Wortlänge und die Anzahl der Stopp-Bits fest (siehe Texteinschub Nr.29). Der ASCII-Wert des zweiten Zeichens gelangt in die Speicherzelle 660 und bestimmt dadurch das Handshake-Protokoll, den Duplex-Modus und die Paritätsprüfung (siehe Texteinschub Nr.29). Zeichen 3 und 4 sind nicht festgelegt.

Man kann den File-Namen des OPEN-Befehls in zwei Arten schreiben, die natürlich identisch sind:

```
(1) OPEN 1,2,0,CHR$(7+64+128) + CHR$(1+16+32)
(2) OPEN 1,2,0,CHR$(199)+CHR$(49)
```

Theoretisch könnte man noch eine dritte Schreibweise hernehmen, nämlich die Zeichen hinschreiben, die den ASCII-Wert 199 beziehungsweise 49 haben. Dann käme die Schreibweise einem File-Namen noch am nächsten. Ein Blick in die Tabelle der ASCII-Codes belehrt uns aber eines Besseren, da wir Zweideutigkeiten nicht ausschließen können. Also ist die Schreibweise der Zeichen mit ihren ASCII-Werten doch am besten. Ich persönlich ziehe die Schreibweise (1) vor, da wir aus ihr sofort die dadurch definierten Werte ablesen können.

»CHR$(7+64+128)«



Sie können die Zusammenhänge direkt der Tabelle 11 entnehmen, die bei der Erklärung der Speicherzelle 659 steht.

Entsprechend wird aus der Tabelle 12 der Speicherzelle 660 das zweite Zeichen »CHR$(1+16+32)« zusammengesetzt:

»CHR$(1+16+32)«

Handshake = X-Leitung
Duplex = Halb-Duplex
Parität = Ungerade

Der OPEN-Befehl mit Gerätenummer 2 hat noch eine Besonderheit, die ich schon bei der Besprechung der Speicherzellen 55 und 56 erwähnt habe. Sobald er nämlich im Programm auftaucht, wird durch ihn der Zeiger in 55 und 56, der ja das obere Ende des Programmspeichers angibt, um 512 Byte nach unten geschoben, um Platz für die beiden Pufferspeicher zu schaffen.

Wenn das mitten im Programm passsiert und vorher schon Zeichenketten (Strings) definiert worden sind (die bekanntlich vom oberen Ende des Speichers aus angelegt werden), werden diese überschrieben.

Also Vorsicht: Wer beabsichtigt, in einem Programm eine RS232-Schnittstelle zu aktivieren, soll diese unbedingt am Anfang des Programms öffnen, damit der Speicherplatz richtig zugeordnet wird.

Daten an den RS232-Kanal übergeben

Die Daten werden zuerst in den Ausgabepuffer gebracht, von dort gelangen sie, vom Handshake-Protokoll gesteuert, an den Empfänger. Die Befehle dazu sind CMD und PRINT#.

»CMD File-Nr.,Zeichen« schaltet vom Bildschirm auf den RS232-Empfänger um.

»PRINT# File-Nr, Zeichen« schreibt die Zeichen in den Ausgabepufferspeicher, von wo sie die Schnittstelle automatisch herausholt.

Beide Befehle wirken genauso wie bei anderen Dateien. Vorsicht ist jedoch geboten, wenn laufend Daten in den Pufferspeicher geschrieben werden, ohne zu wissen, ob die Schnittstelle den Puffer auch wieder entleert hat. Bei Überlauf des Puffers gehen Daten verloren. Es ist ratsam, durch Vergleich der beiden Indizes in den Speicherzellen 669 und 670, die Anfang und Ende des Ausgabepufferspeichers markieren, auf Überlauf zu prüfen.

Daten vom RS232-Kanal übernehmen

Daten, die von der Schnittstelle in den Eingabepufferspeicher gebracht worden sind, werden mit INPUT# oder GET# ausgelesen:

INPUT# File-Nr, Zeichen
GET# File-Nr, Zeichen

Auch hier kann ein Überlaufen des Pufferspeichers auftreten, wenn nämlich die Schnittstelle mehr oder schneller Daten liefert, als mit GET# oder INPUT# ausgelesen werden können. Dieser Zustand kann sowohl durch Überprüfung der Indizes in den Speicherzellen 667 und 668 als auch durch Prüfung von Bit 2 des Statusregisters in 663 erkannt beziehungsweise vermieden werden. Der Speicher kann auch leer sein. Bei Verwendung von INPUT# wartet der Rechner und stürzt bei abgeschalteter Schnittstelle ab. Es ist deshalb empfehlenswert, immer den Befehl GET# zu verwenden, der bei leerem Speicher höchstens einen Nullstring (" ") liefert. Bit 3 des Statusregisters prüft diesen Fall.

Schließen des RS232-Kanals

Der Befehl »CLOSE File-Nr.« schließt den Kanal. Dabei werden die Ein- und Ausgabe-Pufferspeicher aufgelöst, indem der Zeiger in Speicherzelle 55 und 56 wieder auf das Ende des Programmspeichers zeigt. Alle Handshake-Leitungen werden in den Ruhezustand gesetzt und alle Datenübertragungen unterbunden.

Programm-Beispiel

Für eine echte Demonstration müßten Sie eine RS232-Schnittstelle über den User-Port eingerichtet haben. Da ich das nicht voraussetzen kann, begnüge ich mich damit, das im Programmierhandbuch von Commodore angegebene Beispiel zu bringen und zu kommentieren.

```
10 OPEN 1,2,0,CHR$(6+32)+CHR$(32+128)
20 GET#1,A$
30 GET B$
40 IF B$ <> "" THEN PRINT#1,B$;:PRINT B$
50 GET#1,C$
60 PRINT C$;
70 PR = PEEK(663)
80 IF PR=0 OR PR=8 THEN 30
100 IF PR AND 1 THEN PRINT "PARITY-FEHLER"
110 IF PR AND 2 THEN PRINT "BITFOLGE-FEHLER"
120 IF PR AND 4 THEN PRINT "EINGABESPEICHER VOLL"
130 IF PR AND 128 THEN PRINT "UNTERBRECHUNG"
```

Zeile 10 öffnet den RS232-Kanal mit 1 Stopp-Bit, 300 Bit/s und 7 Bit Wortlänge, außerdem über 3-Leitungs-Handshake, Voll-Duplex und ohne Parität.

Zeile 20 will den Eingabespeicher auslesen, der aber noch leer ist. Der resultierende Nullstring interessiert uns nicht, aber die Schnittstelle signalisiert über Handshake, daß wir bereit sind, Daten zu übernehmen.

Zeile 30 fragt inzwischen die Tastatur ab. Wenn eine Taste gedrückt worden ist, schiebt Zeile 40 das Zeichen in den Ausgabespeicher und druckt es nochmal auf dem Bildschirm aus.

Zeile 50 liest wieder den Eingabespeicher aus. Falls inzwischen Daten über die Schnittstelle gekommen sind, druckt Zeile 60 das erste Zeichen auf den Bildschirm.

Zeile 70 ordnet der Variablen PR(üfung) den Inhalt des Statusregisters 663 zu. Ist kein Fehler aufgetreten (PR=0) oder ist der Eingabespeicher immer noch leer (PR=8), dann springt Zeile 80 zurück zur Tastaturabfrage, und der Zyklus läuft weiter. Ist aber ein Fehler aufgetreten, wird dieser ab Zeile 100 geprüft und ausgedruckt.

Adresse 665 und 666 ($299 und $29A)

RS232 – Zeit, die zum Übertragen eines Bits gebraucht wird

Sobald ein RS232-Kanal eröffnet worden ist, berechnet das Betriebssystem einen Wert, der die Zeitdauer eines Bits festlegt. Da die Übertragungsrate in Speicherzelle 659 einstellbar ist, hängt diese Bit-Dauer von der gewählten Übertragungsgeschwindigkeit ab. Die Bit-Dauer errechnet sich aus der Systemfrequenz (985,25 kHz) geteilt durch die Übertragungsgeschwindigkeit. Dieser Wert steht in Low-/High-Byte-Darstellung in diesen beiden Speicherzellen, von wo aus er vom Betriebssystem abgerufen wird.

Adresse 667 ($29B)

RS232-Index auf das Ende des Eingabepufferspeichers

Dieser Index wird verwendet, um Daten in den Eingabepufferspeicher zu schreiben. Wenn man ihn nämlich zum Inhalt der Speicherzelle 247/248 addiert, erhält man die Adresse des zuletzt in den Eingabepufferspeicher eingegebenen Bytes.

Adresse 668 ($29C)

RS232-Index auf den Anfang des Eingabepufferspeichers

Dieser Index wird verwendet, um Daten aus dem Eingabepufferspeicher auszulesen. Wenn man ihn nämlich zum Inhalt der Speicherzelle 247 und 248 addiert, erhält man die Adresse des ersten in den Eingabepufferspeicher eingegebenen Bytes.

Adresse 669 ($29D)

RS232-Index auf den Anfang des Ausgabepufferspeichers

Dieser Index wird verwendet, um Daten aus dem Ausgabepufferspeicher auszulesen. Wenn man ihn nämlich zum Inhalt der Speicherzelle 249 und 250 addiert, erhält man die Adresse des ersten in den Ausgabepufferspeicher eingegebenen Bytes.

Adresse 670 ($29E)

RS232-Index auf das Ende des Ausgabepufferspeichers

Dieser Index wird verwendet, um Daten in den Ausgabepufferspeicher zu schreiben. Wenn man ihn nämlich zum Inhalt der Speicherzelle 249 und 250 addiert, erhält man die Adresse des zuletzt in den Ausgabepufferspeicher eingegebenen Bytes.

Adresse 671 und 672 ($29F und $2A0)

Zwischenspeicher für den IRQ-Vektor während Kassetten-Ein-/Ausgabe

Die Routinen des Betriebssystems, die Daten auf, beziehungsweise von Kassette ein- und ausgeben, werden durch die Interrupt-Routine gesteuert. Diese Routine unterbricht normalerweise 60mal in der Sekunde alle Aktivitäten des Computers, um diverse »Hausaufgaben« (Uhr weiterschalten, STOP-Taste abfragen und so weiter) auszuführen. Bei Kassetten-Ein-/Ausgaben ist diese Interrupt-Routine jedoch abgeschaltet. Dies wird dadurch erreicht, daß der Vektor in Speicherzelle 788 und 789, der auf die Anfangsadresse der Interrupt-Routine zeigt, auf eine Adresse der Kassetten-Routine gesetzt wird. Um nach der Kassettenoperation weitermachen zu können, wird der »alte« Interrupt-Vektor in dieser Speicherzelle 671 und 672 gespeichert.

Adresse 673 ($2A1)

bei C 64: Flagge für RS232-Interrupt bei VC 20: frei verfügbar

Diese Speicherzelle enthält den Wert des Interrupt-Steuerregisters 56589, das die RS232-Schnittstelle steuert. Die Bedeutung der einzelnen Bits, wenn sie auf 1 gesetzt sind, zeigt Tabelle 15. Diese Flagge kann zu Steuerzwecken abgefragt werden. Um beispielsweise ein Programm warten zu lassen, bis der Ausgabepufferspeicher geleert ist, gibt man die Anweisung

```
100 IF (PEEK(673) AND 1)
THEN 100
```

die das Programm so lange aufhält, bis die Übertragung abgeschlossen und Bit 0 der Flagge gelöscht ist.

Die folgenden 4 Speicherzellen, nämlich 674 bis 678, werden nur vom C 64 benutzt. Beim VC 20 sind sie nicht belegt und können frei verwendet werden.

Adresse 674 ($2A2)

Indikator für das Steuerregister A des CIA #1

Mit CIA werden die beiden »Complex Interface Adapter« des C 64 bezeichnet. Das sind integrierte Schaltkreise, die Ein- und Ausgabeoperationen steuern. Jeder der beiden CIAs hat mehrere Register. Das Steuerregister A (Adresse 56334 beziehungsweise $DC0E) beeinflußt die Zählregister des CIA, die ihrerseits die Ein- und Ausgabe von Daten auf beziehungsweise von Kassetten steuern. Das Betriebssystem speichert zu diesem Zweck geeignete Bitmuster in der Speicherzelle 674 ab, die von da in das Steuerregister transferiert werden.

Adresse 675 ($2A3)

Speicher für das Interrupt-Steuerregister B des CIA #1

Ein weiteres Register (Adresse 56333 beziehungsweise $DC0D) ist für die Unterbrechungen (Interrupt) des Computers bei Ein- und Ausgaben zuständig.

In der Speicherzelle 675 werden die Werte dieses Interruptregisters beim Lesen von der Kassette zwischengespeichert.

Bit 0	(Bitwert 1)	Daten werden gesendet
Bit 1	(Bitwert 2)	Daten werden empfangen
Bit 4	(Bitwert 16)	Schnittstelle wartet auf Daten vom Sender

Tabelle 15. Flagge für RS232-Interrupt

Adresse 676 ($2A4)

Zusatzspeicher für Steuerregister B des CIA #1

Derselbe Wert, der bei der Vorbereitung des Lesevorganges von der Kassette in die Speicherzelle 674 kommt, gelangt auch nach 676, von wo er zu einem späteren Zeitpunkt beim Lesen zu Vergleichszwecken herangezogen wird.

Adresse 677 ($2A5)

Zwischenspeicher für das Link-Byte während des Bildschirm-Scrollens

Das Betriebssystem enthält eine Routine, welche den Bildschirminhalt hochschiebt (scrollt), sobald eine leere Zeile eingeschoben wird. Das bedeutet, daß jedesmal die Angaben in den Link-Tabellen der Speicherzellen 217 bis 241 geändert werden müssen. In der Speicherzelle 677 wird nun das Link-Byte zwischengespeichert, während der obere Teil des Bildschirms hochgeschoben wird. Beim VC 20 gibt es diese Funktion übrigens auch. Sie wird durch die Speicherzelle 242 ausgefüllt.

Adresse 678 ($2A6)

Flagge für PAL oder NTSC

Im Gegensatz zum VC 20, der entweder fest auf die deutsche Fernsehnorm PAL oder aber auf die amerikanische Norm NTSC eingestellt ist, kann der C 64 beide Normen verkraften. Diese beiden Normen beziehen sich unter anderem auf die Anzahl der Zeilen und auf die Abtastgeschwindigkeit des Lichtstrahls im Fernsehgerät oder im Monitor. Das Betriebssystem des C 64 überprüft gleich beim Einschalten, ob eine Rasterzeile 311 im angeschlossenen Sichtgerät vorhanden ist. Ist sie nicht vorhanden, muß alles auf die NTSC-Norm eingestellt werden, da diese nur 262 Rasterzeilen hat und mit einer internen Taktfrequenz von 14,3 MHz läuft. Ist eine Rasterzeile 311 vorhanden, wird auf PAL-Norm eingestellt mit einer Taktfrequenz von 17,7 MHz. Das Resultat dieses Tests wird in der Speicherzelle 678 gespeichert: als 0 für NTSC und 1 für PAL.

Adresse 679 bis 767 ($2A7 bis $2FF)

nicht belegt

Diese 89 Byte sind frei und können für alle möglichen Programme und Anwendungen verwendet werden. Beim VC 20 stehen sogar 95 Byte zur Verfügung, da der freie Bereich ja schon ab Speicherzelle 673 beginnt. Dieser Speicherbereich hat den Vorteil, daß er – wie der Kassettenpuffer ja auch – von Basic nicht gestört wird. Er kann also für kleinere Maschinenprogramme oder auch für Sprite-Blöcke verwendet werden. Gegenüber dem Kassettenpuffer hat dieser Bereich den Vorteil, daß er durch Kassettenoperationen nicht gestört wird.

Die nächsten 12 Speicherzellen enthalten 6 Vektoren, deren Bedeutung bei der Übersetzung von Basic-Programmen im Texteinschub Nr. 31 »Indirekte Sprung-Vektoren« näher erklärt wird.

Adresse 768 und 769 ($300 und $301)

Vektor auf die Ausgabe von Fehler-Meldungen (ERROR)

Dieser Vektor zeigt auf die Anfangsadresse der Basic-Routine, welche für die leidigen Fehlermeldungen zuständig ist. Beim C 64 zeigt der Vektor auf 58251 ($E38B), beim VC 20 auf 50234 ($C438).

Diese Routine verwendet eine Tabelle im Basic-Übersetzer, in der alle Fehlermeldungen gespeichert sind. Sie liegt im Speicherbereich 41374 bis 41767 (beim VC 20 49566 bis 49959). Die Routine verwendet den Inhalt des X-Registers (siehe Speicherzelle 781), um die entsprechende Fehlermeldung ganz einfach durch Abzählen der Reihenfolge aus der Tabelle auszulesen und auf dem Bildschirm anzuzeigen.

Ein Verbiegen dieses Vektors ist für zwei Anwendungsfälle sinnvoll. Man kann die Fehlermeldung abschalten, um zu prüfen, ob ein bestimmtes Peripherie-Gerät, zum Beispiel das Floppylaufwerk, angeschlossen beziehungsweise eingeschaltet ist. Die Fehlermeldung ist abschaltbar mit POKE 768,61. Wieder eingeschaltet wird sie mit POKE 768,139. Ein Anwendungsbeispiel habe ich bereits im Texteinschub Nr. 14 »ST-atus« gebracht.

Die zweite Anwendung einer Verbiegung zielt auf eine Übersetzung der Fehlermeldungen. Wem der vorgegebene englische – und manchmal nicht gerade einleuchtende – Text der Fehlermeldungen nicht gefällt, kann den Vektor auf einen Speicherbereich legen, in dem er seine speziellen deutschen Fehlermeldungen speichert. Eine genaue Kenntnis der Fehlermeldungsroutine ist dazu allerdings erforderlich.

Adresse 770 und 771 ($302 und $303)

Vektor auf die Hauptroutine zur Ausführung von Basic-Befehlen

Dieser Vektor zeigt auf die Adresse 42115 ($A483), beim VC 20 auf 50307 ($C483). Die dort beginnende Routine steuert den Direkt-Modus, indem sie entweder direkt eingegebene Befehle ausführt oder mit Zeilennummer eingegebene Anweisungen speichert.

Adresse 772 und 773 ($304 und $305)

Vektor auf die Basic-Routine, die ASCII-Text in Token umwandelt

Dieser Vektor zeigt auf 42364 ($A57C), beim VC 20 auf 50556 ($C57C). Dort beginnt eine Routine, die nach dem Drücken der RETURN-Taste alle Anweisungen der damit eingegebenen Zeile absucht und Text beziehungsweise Wörter, die nicht zwischen Gänsefüßen stehen, als Basic-Befehle interpretiert und sie dann in sogenannte »Token« umwandelt. Token sind Codezahlen, die im Computer anstelle von Textbefehlen verwendet werden. Sie sind im Texteinschub Nr. 32 »Die Kurzschrift von Basic« näher beschrieben.

Dieser Vektor kann verbogen werden, um zusätzliche Basic-Befehle zu erfinden und in das Betriebssystem einzubauen.

Adresse 774 und 775 ($306 und $307)

Vektor auf die Basic-Routine, die Token in ASCII-Werte zurückwandelt (LIST)

Dieser Vektor zeigt auf die Adresse 42778 ($A71A), beim VC 20 auf 50970 ($C71A). Dort beginnt eine Routine, die Token wieder in LISTbaren Text umwandelt. Sie steht nicht allein, sondern wird als Unterprogramm von der LIST-Routine verwendet.

Falls ein Programmierer spezielle zusätzliche Basic-Befehle erfunden hat, kann er durch Verbiegen dieses Vektors seine eigenen Token lesbar ausLISTen.

Man kann auch durch eine entsprechende Verbiegung erreichen, daß die LIST-Routine nicht angesprungen werden kann, was gleichbedeutend ist mit einer LIST-Sperre. Das ist aber wohl nur sinnvoll bei einem Autostart-Programm.

Besser finde ich da ein kleines Programm, das J.Pellechi in der Zeitschrift RUN, Ausgabe 6/85 (Seite 10), angegeben hat:

```
10 FOR J=679 TO 688
20 READ K
30 POKE J,K
40 NEXT J
50 POKE 774,167:POKE 775,2
60 NEW
70 DATA 72,173,141
80 DATA 2,208,251,104
90 DATA 76,26,167
```

Beim VC 20 ist nur die Zeile 90 verschieden:

```
90 DATA 76,26,199
```

In den freien Speicherbereich ab Speicherzelle 679 wird ein kleines Maschinenprogramm gePOKEt, das in den DATA-Zeilen 70 bis 90 steht. In Zeile 50 steht der für unser Beispiel entscheidende Befehl: Der Vektor in 774 und 775 wird nach der Adresse 679 verbogen. Dadurch springt die LIST-Routine immer zuerst auf die Adresse 679, in der sie das kleine Maschinenprogramm findet.

Disassembliert schaut das so aus:

```
02A7 48         PHA
02A8 AD 8D 02   LDA 028D
02AB D0 FB      BNE 02A8
02AD 68         PLA
02AE 4C 1A A7   JMP A71A
```

Zuerst wird der Akkumulator mit dem Inhalt der Speicherzelle 653 ($28D) geladen. Dort steht bekanntlich eine Zahl von 1 bis 7, je nachdem, ob die SHIFT-, CTRL- oder Commodore-Taste gedrückt ist. Ist dies der Fall, springt das Programm auf die Adresse 680 zurück und bildet so eine Dauerschleife, bis die Taste wieder losgelassen wird. Erst dann geht es weiter mit der ursprünglichen Zieladresse des Vektors in 774 und 775, nämlich $A71A (42778) beziehungsweise $C71A (50970) beim VC 20.

Auf diese Weise können Sie das LISTen eines Programms mit einer der drei genannten Tasten anhalten.

Adresse 776 und 777 ($308 und $309)

Vektor auf die Basic-Routine, die den nächsten Befehl liest und ausführt

Dieser Vektor zeigt auf die Adresse 42980 ($A7E4), beim VC 20 auf 51172 ($C7E4). Diese Routine prüft das nächste Token, ob es gültig ist. Wenn der ASCII-Wert des Token kleiner als 128 ist, wird er als Zeichen einer Variablen angesehen, und das System springt auf die LET-Routine. Das erklärt, warum zur Definition einer Variablen der LET-Befehl auch weggelassen werden kann.

Durch Verbiegen dieses Vektors kann zum Beispiel eine Trace-Routine gebaut werden, welche zuerst die Nummer der Zeile ausdruckt, die gerade ausgeführt wird, bevor sie auf die ursprüngliche Zieladresse des Vektors zurückkehrt.

Adresse 778 und 779 ($30A und $30B)

Vektor auf die Basic-Routine, die einen numerischen Ausdruck in eine Gleitkommazahl umwandelt

Dieser Vektor zeigt auf 44675 ($AE83), beim VC 20 auf 52867 ($CE83). Hier beginnt eine Routine, die einen einzelnen numerischen Wert, wenn er Teil eines Ausdrucks ist, von seinem ASCII-Wert in eine Gleitkomma-Zahl umwandelt.

Ist der Ausdruck eine Konstante, wird diese Umwandlung durchgeführt.

Ist der Ausdruck eine Variable, wird ihr Zahlenwert aus dem Variablenspeicher geholt.

Ist der Ausdruck die Zahl »pi«, wird der Zahlenwert für »pi« in den Gleitkomma-Akkumulator gebracht.

Speicher für die Register des Mikroprozessors

Der SYS-Befehl holt aus den nächsten vier Speicherzellen alle notwendigen Parameter, die für ein mit SYS zu startendes Maschinenprogramm notwendig sind. Er speichert sie in die vier Register des Mikroprozessors 6510 (beim VC 20 heißt er 6502). Es sind dies:

- der Akkumulator
- das X-Register

- das Y-Register
- das P-(Status-)Register

Die Bedeutung der Register ist im Assembler-Kurs erklärt worden.

Normalerweise funktioniert der SYS-Befehl nur, wenn vorher schon alle Parameter des aufgerufenen Maschinenprogramms richtig vorhanden sind, was meistens nicht der Fall ist.

So können Sie zum Beispiel mit Aufrufen der Load-Routine durch SYS 62622 nichts ausrichten, weil die für LOAD erforderlichen Parameter, nämlich Gerätenummer, File-Namen, Anfangs- und Endadresse, nicht festgelegt sind.

Wie dies mit Hilfe der vier folgenden Register-Speicherzellen erreichbar ist, hat Rolf Zweifel schon in der Ausgabe 7/84, Seite 131 erklärt. Weil das aber schon lange her ist und weil es hier so schön in den Kurs paßt, wiederhole ich dieses Thema im Texteinschub Nr.33 »Der vorbereitete SYS-Befehl«.

Adresse 780 ($30C)

Speicher für den Akkumulator

Adresse 781 ($30D)

Speicher für das X-Register

Adresse 782 ($30E)

Speicher für das Y-Register

Adresse 783 ($30F)

Speicher für das Statusregister

Die nächsten drei Speicherzellen 784 bis 786 sind beim VC 20 nicht belegt. Beim C 64 entsprechen sie den Adressen 0 bis 2 des VC 20.

Adresse 784 bis 786 ($310 bis $312)

Sprungbefehl und wählbare Sprungadresse des USR-Befehls

Mit dem Basic-Befehl USR wird bekanntlich ein Maschinenprogramm gestartet. Diese drei Speicherzellen werden bei der Abwicklung von USR verwendet. In ihnen muß der Anwender des USR-Befehls die Zieladresse in Low-/High-Byte-Darstellung angeben, ab der das Maschinenprogramm im Speicher steht.

Dieser Vorgang ist bereits behandelt worden bei den Speicherzellen 0 bis 2 des VC 20, die ja genau den Speicherzellen 784 bis 786 des C 64 entsprechen.

Speziell für den C 64 ist der USR-Befehl noch einmal behandelt, und zwar im Texteinschub Nr. 34 »Das Mauerblümchen USR«.

Adresse 787 ($313)

beim C 64 und VC 20 nicht belegt

Während dieses freie Byte des C 64 nicht viel nutzt, haben VC 20-Besitzer immerhin vier aufeinanderfolgende freie Byte für eigene Vektoren und andere zwischenzuspeichernde Werte zur Verfügung, die nie in Gefahr geraten, von einem Basic-Programm überschrieben zu werden.

Texteinschub Nr. 31 Indirekte Sprung-Vektoren

Mit »Vektor« wird ein Adressenpaar bezeichnet, dessen Inhalt in der Low-/High-Byte-Darstellung wiederum eine Adresse angibt, ab der ein Maschinenprogramm beginnt.

Wenn man nun mit dem Maschinencode-Befehl JMP (jump) auf die Adresse springt, die der Vektor angibt, läuft das Maschinenprogramm ab dieser Adresse los.

Bekanntlich stehen im nicht veränderbaren ROM-Speicher viele Unterprogramme (Routinen) des Basic-Übersetzers und des Betriebssystems, die auch für andere Programme verwendbar sind. Commodore hat nun die brillante Idee gehabt, mehrere dieser Routinen herauszusuchen und ihre Anfangsadressen zur leichten Verwendung benutzerfreundlich in einer Tabelle zusammenzustellen, wo sie mit dem schon genannten Sprungbefehl »angewählt« werden können.

Diese Tabelle ist deshalb interessant, weil die Anfangsadressen der Routinen bei den einzelnen Commodore-Computern verschieden sind, obwohl sie eigentlich fast identische Übersetzer und Betriebssysteme haben. So beginnt zum Beispiel die LOAD-Routine des C 64 ab Adresse 62622 ($F49E), beim VC 20 aber ab Adresse 62786 ($F542).

Um zu erreichen, daß Programme, die diese Routinen benutzen, trotzdem von einem Commodore-Computer auf einen anderen übertragbar sind, hat Commodore diese Tabelle geschaffen, welche den Sprung auf diese Routinen unabhängig vom Computertyp macht.

Sie liegt (bei allen Commodore-Typen) im Bereich 768 bis 779 für Routinen des Basic-Übersetzers und 788 bis 819 für Routinen des Betriebssystems.

Diesen Zusammenhang zeige ich aber besser an einem Beispiel:
Der Vektor auf die LOAD-Routine hat die Adresse 816/817.
Wir schauen nach, was dort steht:

```
PRINT PEEK(816) PEEK(817)
```

Wir erhalten beim C 64 die Zahlen 158 und 244.
Beim VC 20 lautet das Ergebnis 66 und 245
Beide Zahlenpaare werden nach der üblichen Low-/High-Byte Methode umgerechnet:
158 + 256 * 244 ergibt 62622.
66 + 256 * 245 ergibt 62786.

Das sind aber genau die weiter oben schon genannten Anfangsadressen der LOAD-Routine im Betriebssystem. Mit einem Sprung auf 816 und 817 landet ein Maschinenprogramm also immer zwangsläufig auf der LOAD-Routine. Kenner wissen, daß ein derartiger Sprung »indirekt« sein muß, also mit dem Befehls-Code $6C, der nicht auf die angegebene Adresse, sondern auf die in ihr enthaltene Zieladresse springt.

Diese indirekte Sprungmethode hat außer der schon erwähnten Unabhängigkeit vom Computertyp noch einen weiteren Vorteil:

Da der Vektor, der auf die Zieladresse zeigt, im RAM-Speicher liegt, kann er verändert werden. Das bedeutet, daß dem Programmierer die Möglichkeit geboten wird, in ursprünglich »festgefrorene« Routinen des Übersetzers (Interpreter) und des Betriebssystems beliebige Änderungen und Varianten einzubauen. Ich will Ihnen das mit einem zwar nutzlosen, aber dennoch verblüffenden Beispiel zeigen. Bekanntlich meldet sich der Computer nach dem Befehl LOAD mit der Anweisung »PRESS PLAY ON TAPE«, weil der Vektor in 816 und 817 auf die Routine zeigt, die den LOAD-Befehl ausführt.

Jetzt verbiegen wir den Vektor so, daß er auf die SAVE-Routine zeigt. Diese Routine beginnt ab Speicherzelle ($E156) – beim VC 20 ab 57683 ($E153). Diese Adresse POKEn wir in Low-/High-Byte-Darstellung nach 816 und 817.

```
POKE 816,86:POKE 817,225 (VC 20:POKE 816,83)
```

Wenn Sie jetzt LOAD eingeben, meldet sich der Computer mit »PRESS RECORD AND PLAY ON TAPE«, der Anweisung für SAVE.

Weitere Beispiele dafür finden Sie im Kurs bei den entsprechenden Speicherzellen. Diese Art der Modifikation entspricht der oft genannten Wedge-Methode, die auch ich in diesem Kurs erwähnt habe, und zwar bei der Besprechung der Speicherzellen 115 bis 138. Aber der Einsatz der indirekten Sprung-Vektoren ist halt viel eleganter.

Texteinschub Nr. 32 Die Kurzschrift von Basic

Immer wenn Sie eine Anweisung auf den Bildschirm schreiben, sei es als Programmzeile, sei es als Direkt-Eingabe, wird sie in den Tastaturpuffer gebracht. Sobald die Eingabe mit der RETURN-Taste abgeschlossen wird, werden die Anweisungen dieser Zeile entweder in den Programmspeicher gebracht oder aber direkt ausgeführt. Beides geschieht allerdings nicht sofort, da der Computer ja bekanntlich intern nicht mit Buchstaben und Dezimalziffern, sondern mit besonderen Codezahlen arbeitet. Deshalb wird der Text einer Zeile zuerst in diese Codewerte umgewandelt und in eine besondere Reihenfolge gebracht. Ziffern, Zeichen und Texte, die innerhalb von Gänsefüßen stehen,

werden mit ihrem ASCII-Code abgespeichert. Basic-Befehle und Basic-Funktionen werden in Zahlen umgewandelt. Diese heißen in der Fachsprache »TOKEN«.

Der Befehl PRINT wird also nicht als Folge von 5 ASCII-Werten, sondern als einzelnes Byte mit dem Wert 153 gespeichert. Da meines Wissens die Liste der Token nur im Programmierhandbuch des VC 20 und im 64'er, Ausgabe 9/84 im Kurs von Christoph Sauer veröffentlicht ist, gebe ich in der weiter unten folgenden Tabelle alle Werte noch einmal an.

Bemerkenswert ist, daß für GET# kein eigener Token vorhanden ist, dafür aber einer (203) für einen Befehl, den es in den Handbüchern nicht gibt, nämlich GO. Das erklärt übrigens, warum der Befehl GOTO auch in der Form GO TO geschrieben werden darf. Die Routine, welche Basic-Befehle ausführt, erkennt nämlich die beiden Token an und kombiniert sie miteinander.

Interessant ist auch, daß die Befehle TAB und SPC in ihren Token gleich die linke Klammer mit einschließen. Nach dieser Tabelle sieht eine im Programmspeicher gespeicherte Zeile so aus:

```
10 IF A=5THEN PRINT TAB(X)
```

Link	10	0	139	65	178	53	167	32	153	32	163	88	41	0
	Zeil.-Nr.		IF	A	=	5	THEN		PRINT		TAB(	X	)	

Diese Zahlen können Sie selbst überprüfen, indem Sie diese Zeile eingeben und dann den Anfang des Programmspeichers sichtbar machen:

```
FOR J=1 TO 20:PRINT PEEK(2048+J);:NEXT
```

Beim VC 20 müssen Sie den von der Speichererweiterung abhängigen Anfang des Programmspeichers einsetzen. Sie werden dieselbe Zahlenreihe wie oben erhalten.

Die Technik, in einem Programm direkt die Token anstelle von Basic-Befehlswörtern zu verwenden, bieten dem Programmierer in Maschinensprache eine gute Möglichkeit, Speicherplatz zu sparen. Das kann insbesondere bei großen Textprogrammen, wie zum Beispiel bei Adventure-Spielen, nützlich sein. Der Vollständigkeit halber muß ich noch erwähnen, daß die Token bei dem LIST-Befehl wieder in ihre ursprüngliche Textform zurückgewandelt werden. Die Vektoren für die Wandel- beziehungsweise Rückwandel-Routinen stehen in 772 und 773 und 774 und 775.

128 END	154 CONT	180 SGN
129 FOR	155 LIST	181 INT
130 NEXT	156 CLR	182 ABS
131 DATA	157 CMD	183 USR
132 INPUT#	158 SYS	184 FRE
133 INPUT	159 OPEN	185 POS
134 DIM	160 CLOSE	186 SQR
135 READ	161 GET	187 RND
136 LET	162 NEW	188 LOG
137 GOTO	163 TAB(	189 EXP
138 RUN	164 TO	190 COS
139 IF	165 FN	191 SIN
140 RESTORE	166 SPC(	192 TAN
141 GOSUB	167 THEN	193 ATN
142 RETURN	168 NOT	194 PEEK
143 REM	169 STEP	195 LEN
144 STOP	170 +	196 STR$
145 ON	171 -	197 VAL
146 WAIT	172 *	198 ASC
147 LOAD	173 /	199 CHR$
148 SAVE	174 ↑	200 LEFT$
149 VERIFY	175 AND	201 RIGHT$
150 DEF	176 OR	202 MID$
151 POKE	177 größer	203 GO
152 PRINT#	178 =	
153 PRINT	179 kleiner	

Texteinschub Nr. 33
Der vorbereitete SYS-Befehl

Programme, die in Maschinensprache geschrieben sind, können von einem Basic-Programm aus mit dem SYS-Befehl angewählt und ausgeführt werden.

Im Prinzip gilt das auch für Routinen des Basic-Übersetzers (Interpreter) und des Betriebssystems, die fest im ROM-Speicher untergebracht sind.

Ein Beispiel dafür ist SYS 58260, der Sprung auf den Kaltstart – beim VC 20 ist es SYS 58232, der den Computer in die Ausgangslage zurücksetzt.

Die meisten dieser Routinen benötigen jedoch verschiedene Angaben – man nennt sie auch Parameter – die vor der Ausführung des SYS-Befehls richtig eingestellt sein müssen.

Die LOAD-Routine zum Beispiel, die ab Speicherzelle 62622 ($F49E) – beim VC 20 ab 62793 ($F549) – beginnt, können wir mit dem SYS 62622 nicht starten. Es fehlen die Angaben über Geräte-Nummer (8 für Floppy, 1 für Band), File-Namen, sowie Anfangs- und Endadresse. Diese Parameter werden normalerweise nach dem Befehl LOAD von der Routine des Interpreters, die den LOAD-Befehl übersetzt, eingegeben. Wir geben ja nicht einfach LOAD ein, wenn wir ein Programm mit dem Namen »Test« auf Diskette speichern wollen, sondern wir schreiben

```
LOAD "TEST",8.
```

Auch wenn wir nur LOAD eintippen, werden vom Übersetzer Parameter gesetzt, nämlich »namenlos« und 1 für Bandgerät. Ich hoffe, Ihnen ist geläufig, daß beim Weglassen aller Angaben der Übersetzer immer Kassettenoperationen durchführt. Natürlich können wir uns das anschauen:

Die Routine des Übersetzers für den Basic-Befehl LOAD beginnt an Speicherzelle 57704 ($E168), beim VC 20 bei 57700 ($E164).

Mit SYS 57704 springen wir dorthin – und in der Tat, wir erhalten auf dem Bildschirm »PRESS PLAY ON TAPE«. Aber ein Programm auf diese Weise von der Floppy zu LOADen, gelingt uns nicht, es sei denn, wir können die fehlenden Parameter von Hand eingeben.

Genau das aber können wir, weil der SYS-Befehl sich diese Parameter aus den Speicherzellen 780 bis 783 holt und in die vier Register des Mikroprozessors schreibt.

780 ist die Adresse des Akkumulators
781 ist die Adresse des X-Registers
782 ist die Adresse des Y-Registers
783 ist die Adresse des Status-Registers.

Die Behandlung von A, X und Y ist unkompliziert, wie wir gleich sehen werden.

Das Status-Register, manchmal auch P-Register genannt, ist nicht so einfach zu verwenden, da es nicht Zahlenwerte, sondern Flaggen (Bitmuster) enthält. Im einzelnen bedeuten:

BIT Nr.	WERT	FLAGGE	ABKÜRZUNG
0	1	Übertrag	C(arry)
1	2	NULL	Z(ero)
2	4	Unterbrechung	I(nterrupt)
3	8	Dezimal	D
4	16	Abbruch	B(reak)
5	32	nicht benutzt	
6	64	Überlauf	V
7	128	Vorzeichen	N(egativ)

Um eine der Flaggen des Status-Registers zu löschen, empfiehlt es sich, das ganze Register mit POKE 783,0 zu löschen. Umgekehrt muß man beim Setzen der Bits sehr aufpassen wegen der Unterbrechungsflagge I. Eine 1 in I entspricht dem Maschinenbefehl SEI, der alle Interrupts ausschaltet, auch die der Tastatur-Abfrage, was natürlich sehr störend sein kann! Um alle Flaggen außer der Unterbrechungsflagge I zu setzen, muß POKE 783,247 eingegeben werden.

So, jetzt wird es Zeit für ein Beispiel, wie vor dem SYS-Befehl Parameter eingegeben werden können. In der Literatur wird immer das Beispiel gewählt, den Cursor auf eine bestimmte Posi-

tion zu setzen, beziehungsweise seine Position abzufragen. Dazu gibt es eine Routine, die bei beiden Computern ab Speicherzelle 65520 ($FFF0) beginnt.

Sie nimmt die Zahl, die im X-Register steht, und verwendet sie als Zeilennummer; die Zahl des Y-Registers nimmt sie als Spaltennummer, setzt dann den Cursor an diese Stelle und bringt die beiden Werte in die Speicherzellen 209 und 210 und 211.

Unser Beispiel hat die Aufgabe, den Cursor in die vierte Spalte der siebten Zeile zu setzen, dort das Dollar-Zeichen hinzuschreiben und es rot zu färben.

```
 5 PRINT CHR$(147)
10 POKE 783,0
20 POKE 781,6
30 POKE 782,3
40 SYS 65520
```

Nach Löschen des Bildschirms werden zuerst alle Flaggen des Statusregisters gelöscht (Zeile 5). Dann kommt die Zeilennummer in das X-Register (Zeile 10) und die Spaltennummer in das Y-Register (Zeile 30). Nach dem Eingeben dieser Parameter können wir mit SYS auf die Routine springen.

```
50 ZEILE=PEEK(209)+256*PEEK(210)
60 ADRESSE = ZEILE + PEEK(211)
70 POKE ADRESSE,36
```

In Speicherzellen 209/210 können wir jetzt (zur Übung) die Zeilennummer ablesen. Die Adresse der Cursorposition im Bildschirmspeicher erhalten wir durch die Addition der Zeilennummer mit dem Inhalt der Speicherzelle 211. Dorthin POKEn wir den Bildschirmcode des Dollarzeichens, nämlich 36 (Zeile 70).

```
 80 SYS 59940
 90 FARBE=PEEK(243)+256*PEEK(244)
100 POKE FARBE+PEEK(211),2
```

Für das Färben des Dollarzeichens verwenden wir eine weitere Routine des Betriebssystems, die ab 59940 – beim VC 20 ab 60082 – beginnt. Sie ermittelt die Zeilenposition des Cursors im Farbspeicher und bringt diesen Wert in die Speicherzellen 243 und 244, wo wir ihn abfragen können (Zeile 90). Die Adresse der Cursorposition im Farbspeicher setzt sich aus diesem Wert plus der Spaltennummer zusammen, die wir wieder der Speicherzelle 211 entnehmen. Auf diesen Platz POKEn wir den Farbcode 2 für rot (Zeile 100). So leicht ist das, wenn man die Routinen und die Aufgaben der Speicherzellen kennt.

Die letzteren lernen Sie in diesem Kurs. Die Beschreibung und Anwendung der Routinen muß, wie schon öfters erwähnt, einem eigenen Kurs vorbehalten bleiben.

Texteinschub Nr. 34
Das Mauerblümchen USR

Hand aufs Herz: Haben Sie den USR-Befehl schon einmal benutzt? Ohne Zweifel gehört er zu den Mauerblümchen von Basic, obwohl sein Name – eine Abkürzung von USER (Verwender) – eigentlich genügend Anreiz bieten müßte. Da er ohne die Speicherzellen 784 bis 786 nicht auskommt, ist dieser Teil des Kurses eine gute Gelegenheit, ihn Ihnen näherzubringen.

USR hat im Grunde genommen dieselbe Funktion wie SYS. Er springt nämlich aus einem Basic-Programm direkt in ein Maschinen-Programm, arbeitet dieses so lange ab, bis er den Befehl RTS findet. RTS entspricht dem Basic-Befehl RETURN und springt in das Basic-Programm zurück.

Bei SYS steht die Sprungadresse gleich hinter dem Befehl.

Bei USR muß die Sprungadresse zuerst in die Speicherzellen 785 und 786 gePOKEt werden (beim VC 20 in 1 und 2).

Beispiel: Sprung auf 56524 ($DCCC)

mit SYS: `SYS 56524`

mit USR: `POKE 785,204 (204+256*220=56524) POKE 786,220`
`X=USR(Y)`

Kein Wunder, daß USR selten verwendet wird – ist er doch durch das POKEn der Sprungadresse in Low-/High-Byte-Darstellung aufgebläht.

Das ist aber nicht unnütz, weil USR mehr Fähigkeiten hat als SYS. Im Hinblick auf die im anderen Texteinschub Nr. 33 »Der vorbereitete SYS-Befehl« aufgezeigten Möglichkeiten des SYS-Befehls sollte ich besser sagen: USR hat andere Fähigkeiten als SYS.

USR ist eine Mischung von SYS und FN. Letzterer ist der Basic-Befehl zur Definition selbst erfundener Funktionen. Bei USR allerdings wird die Funktion als Unterprogramm in Maschinensprache geschrieben, auf die dann wie gesagt der USR-Befehl zur Ausführung springt. Der Pfiff dabei ist aber, daß Zahlenwerte in das Maschinenprogramm mitgenommen beziehungsweise Resultate aus ihm herausgeholt werden können.

Wie läuft das ab: Das Argument Y, das in der Klammer hinter dem Befehl steht, wird zuerst in den Gleitkomma-Akkumulator Nr. 1 (FAC 1) in den Speicherzellen 97 bis 102 gebracht. Als Gleitkommazahl wird es vom angesprungenen Maschinenprogramm weiterverarbeitet. Das Resultat kommt dann wieder in den FAC 1 und steht als Wert von X zur Verfügung.

Das Argument Y kann übrigens auch ein komplexer Ausdruck sein, zum Beispiel: `X=USR(PEEK(A)+256*PEEK(B))`

Ich möchte das an einem kleinen Beispiel demonstrieren.

Statt allerdings ein Maschinenprogramm selbst zu schreiben, verwende beziehungsweise springe ich auf eine Routine des Betriebssystems, die den Inhalt des FAC 1 für mathematische Operationen verwendet.

Als geeignete mathematische Operation habe ich die Routine für die Funktion INT gewählt, die im C 64 ab der Adresse 48332 ($BCCC) beginnt, beim VC 20 ab 56524 ($DCCC).

Zuerst definieren wir einen Wert für die Variable Y, der in die INT-Routine gebracht werden soll:

```
10 Y=14,35
```

Dann bestimmen wir die Sprungadresse für den USR-Befehl. Dazu teilen wir die Adresse 48332 auf in ein Low-Byte = 204 und ein High-Byte = 188. Bei der Startadresse 56524 von INT im VC 20 sind dies 204 und 220. Diese POKEn wir nach 785 und 786 beziehungsweise nach 1 und 2:

```
20 POKE 785,204
30 POKE 786,188
```

Jetzt folgt nur noch der USR-Befehl selbst und das Ausdrucken des Resultats.

```
40 X=USR(Y)
50 PRINT X
```

Beim VC 20 lauten die Zeilen 20 und 30:

```
20 POKE 1,204
30 POKE 2,220
```

Nach RUN erhalten wir das Resultat 14, wie das Gesetz für INT es befiehlt.

Sie können zur Übung statt INT auch COS verwenden, indem Sie auf die Adresse 57938 ($E264) beziehungsweise beim VC 20 auf 57935 ($E261) springen. Der Vergleich mit dem Befehl COS Y muß dasselbe Ergebnis bringen. Wer hat übrigens gemerkt, daß wir überhaupt nichts mit der Speicherzelle 784 (beziehungsweise 0) gemacht haben, obwohl sie doch angeblich am USR-Befehl beteiligt ist?

Sie ist es wirklich, doch ohne unser Zutun. In diese Adresse wird beim Einschalten des Computers die Zahl 76 ($4C) geschrieben. Das ist der Code für den Maschinenbefehl »JMP« (JUMP), der dieselbe Wirkung hat wie GOSUB.

Bei Ausführung von USR springt nämlich die entsprechende Routine zuerst auf die Speicherzelle 784 (beziehungsweise 0), findet dort den Sprungbefehl und in den beiden nachfolgenden Speicherzellen 785 und 786 (beziehungsweise 1 und 2) die Sprungadresse – und führt so den geplanten Sprung aus.

Ich finde, USR ist es wert, in Ihre Überlegungen mit einbezogen zu werden, besonders wenn Sie innerhalb Ihrer Basic-Programme extrem schnelle Unterprogramme in Maschinensprache eingebaut haben. Diese sind mit USR ganz elegant aufrufbar. Ich denke da zum Beispiel an eine Abfrage der Joysticks oder der Paddle.

Adresse 788 und 789 ($314 und $315)

Vektor auf die IRQ-Interrupt-Routine des Betriebssystems

Dieser Vektor zeigt auf die Adresse 59953 ($EA31) – beim VC 20 auf 60095 ($EABF). Ab hier beginnt die Routine des Betriebssystems, die den IRQ-Interrupt ausführt. Die Bedeutung der verschiedenen Interrupts (Unterbrechungen), ihre Auslöser und Abläufe sind als Übersicht im Texteinschub Nr. 35 »Dem Computer ins Wort fallen« dargestellt.

Die IRQ-Routine wird vom Timer A des Ein-/Ausgabe-Bausteins CIA #1 – beim VC 20 vom Timer 1 des Ein-/Ausgabe-Bausteins VIA #2 – ausgelöst, und zwar periodisch 60mal in jeder Sekunde. In der Programmpause werden die im Texteinschub beschriebenen »Haushaltsarbeiten« durchgeführt.

Dieser Vektor eignet sich hervorragend für eigene Programmierzwecke, da er durch das Verbiegen auf eine andere Adresse seine gleichmäßige und hochfrequente Wiederkehr nicht verliert. Mit seiner Hilfe können also eigene Maschinenprogramme 60mal in der Sekunde in ein Programm eingeschoben werden – eine Methode, die deswegen den englischen Namen »Wedge« = Keil, erhalten hat. Zwei Vorbedingungen sind allerdings dabei zu erfüllen.

1. Da ein IRQ mit Sicherheit während des Verbiegens auftritt, muß er vorher abgeschaltet werden. Den Schlüssel dazu bietet die Speicherzelle 56334, die mit 0 gePOKEt den Interrupt abschaltet und mit POKE 56334,1 ihn wieder zuläßt. Beim VC 20 ist dies POKE 37116,127 beziehungsweise POKE 37116,192. Aber Vorsicht!! Da während eines IRQ-Interrupts auch die Tastatur abgefragt wird, kann das Abschalten nur innerhalb eines Programms erfolgen – während der Abschaltung ist die Tastatur tot.

2. Am Ende eines »Wedge« muß der Sprung auf die alte IRQ-Adresse erfolgen, die ursprünglich in den Speicherzellen 788 bis 789 stand, damit – etwas verspätet zwar – die normalen Haushaltsarbeiten des IRQ nachgeholt werden können. Bei längeren Wedges wird daher die interne Uhr TI und TI$ etwas nachgehen.

Ich habe lange nach einem Beispiel gesucht. Ich kenne viele: Abfrage der Joysticks, Lautstärke von Tönen mit Funktionstasten steuern, von Basic unabhängige Laufschrift, um ein paar zu nennen. Aber alle haben einen ziemlich langen Maschinensprache-Teil. Ich bringe daher hier das kürzeste Beispiel, das ich kenne. Es stammt von Rügheimer und Spanik.

Das Programm verändert dauernd die Farbe des Bildschirmrahmens:

```
10 FOR K=679 TO 699
20 READ A
30 POKE K,A:NEXT
40 DATA 166,162,224,0,
   224,128,240
50 DATA 3,76,49,234,174,
   32,208
60 DATA 202,142,32,208,
   76,175,02
70 POKE 56334,0
80 POKE 788,167:
   POKE 789,2
90 POKE 56334,1
```

Dieses Programm gilt nur für den C 64; für den VC 20 müßte es entsprechend umgeschrieben werden.

Die Zeilen 10 bis 30 lesen das Maschinenprogramm, das in den DATA-Zeilen 40 bis 60 steht, in die Speicherzellen 679 bis 699. Diese stehen, wie wir das letzte Mal gesehen haben, zur freien Verfügung – und sind daher ideal geeignet, ein kleines Maschinenprogramm ungestört aufzunehmen.

In Zeile 70 wird der IRQ-Interrupt unterbrochen. Jetzt kommt der wichtige Teil: Zeile 80 verbiegt den IRQ-Vektor zur Speicherzelle 176 + 256 * 2 = 679. Zeile 90 schaltet schließlich den IRQ-Interrupt wieder ein.

Jetzt passiert also folgendes: Jedesmal, wenn der Timer A den Haushalt-IRQ auslöst, springt der Computer zuerst einmal auf das Maschinenprogramm ab Speicherzelle 679 und schaltet die Rahmenfarbe um. Dann erst springt der letzte Befehl des Maschinenprogramms auf die ursprüngliche IRQ-Adresse 59953 ($EA31), von der aus das Betriebssystem weitermacht, als sei nichts geschehen.

Für Kenner gebe ich noch das Assembler-Listing des Maschinenprogramms an:

```
,02A7 A6 A2    LDX A2
,02A9 E0 00    CPX #00
,02AB E0 80    CPX #80
,02AD F0 03    BEQ 02B2
,02AF 4C 31 EA JMP EA31
,02B2 AE 20 D0 LDX D020
,02B5 CA       DEX
,02B6 8E 20 D0 STX D020
,02B9 4c Af 02 JMP 02AF
```

Es gibt noch eine kleine, erwähnenswerte Anwendung. Wenn der Vektor nicht auf den Anfang der IRQ-Routine bei 59953, sondern auf 59956 – also drei Stellen weiter – zeigt, übergeht die IRQ-Routine den Teil, welcher die STOP-Taste abfragt und die TI/TI$-Uhr weiterschaltet, wodurch effektiv die STOP-Taste ausgeschaltet wird. Das geht ganz schnell mit POKE 788,52. Mit POKE 788,49 wird das wieder rückgängig gemacht. Beim VC 20 sind es die Werte POKE 788,194 oder POKE 788,191.

Adresse 790 und 791 ($316 und $317)

Vektor auf die BREAK-Interrupt-Routine des Betriebssystems

Diese Routine ist im Texteinschub Nr. 35 nicht erwähnt, weil sie ein Teil der NMI-Routine ist. Dieser Vektor zeigt auf die Adresse 65126 ($FE66) – beim VC 20 auf 65234 ($FED2). Die da beginnende Routine des Betriebssystems wird aufgerufen, wenn der Maschinenbefehl BRK ausgeführt wird. Er führt letztlich zu einem Warmstart, das heißt der Bildschirm wird gelöscht und der Cursor meldet sich mit READY. Diese Routine wird auch durch das gleichzeitige Drücken der STOP- und der RESTORE-Taste angestoßen.

Adresse 792 und 793 ($318 und $319)

Vektor auf die NMI-Routine des Betriebssystems.

Der NMI-Interrupt ist im Texteinschub Nr. 35 »Dem Computer ins Wort fallen« näher beschrieben. Der Vektor zeigt auf den Beginn dieser Routine ab Speicherzelle 65095 ($FE47) – beim VC 20 ab 65197 ($FEAD).

Sobald ein NMI-Interrupt auftritt, wird zuerst durch Setzen der Interrupt-Abschalt-Flagge (Interrupt Disable Flag) jede Unterbrechung durch den IRQ-Interrupt unterbunden. Dann wird geprüft, wer den NMI-Interrupt ausgelöst hat, und zwar in der Reihenfolge: RS232-Schnittstelle, RESTORE-Taste; eingestecktes Modul und schließlich die STOP-Taste. Die letztere dient zum Sichern der RESTORE-Taste. Nur wenn beide gemeinsam gedrückt werden, kommt die NMI-Unterbrechung durch die RESTORE-Taste zur Auswirkung.

Da die RESTORE-Taste fast als erste abgefragt wird, kann sie und ihre Kombination mit der STOP-Taste durch Verbiegen des Vektors in Speicherzelle 792 bis 793 abgeschaltet werden. Beim C 64 geht das mit POKE 792,193. Wieder eingeschaltet wird mit POKE 792,71. Beim VC 20 geht das mit POKE 792,91 beziehungsweise POKE 792,173. Natürlich können Spezialisten durch Verbiegen des Vektors auf andere Adressen ihre eigenen NMI-Routinen bauen.

Adresse 794 und 795 ($31A und $31B)

Vektor auf die OPEN-Routine des Betriebssystems

Die Routine beginnt ab Adresse 62282 ($F34A) – beim VC 20 ab 62474 ($FEAD). Diese Routine prüft, ob eine Datei (File) eröffnet werden kann. Das geht immer dann, wenn die File-Nummer nicht 0 ist und wenn weniger als 10 andere Dateien bereits eröffnet sind. Für die serielle Schnittstelle (Geräte-Nummer 4, 5, 8 bis 11) wird an das angewählte Gerät zuerst der Befehl »Listen« gegeben und dann die Sekundär-Adresse des OPEN-Befehls.

Beim Bandgerät (Geräte-Nummer 1) prüft die Routine den Tape Header einer sequentiellen Datei beziehungsweise schreibt einen Tape Header auf das Band.

Bei Anwahl der RS232-Schnittstelle (Geräte-Nummer 2) aktiviert die Routine einige Leitungen und reserviert je einen Ein- und Ausgabe-Pufferspeicher am oberen Ende des Basic-Programmspeichers.

Adresse 796 und 797 ($31C und $31D)

Vektor auf die CLOSE-Routine des Betriebssystems

Dieser Vektor zeigt auf die Adresse 62097 ($F291) – beim VC 20 auf 62282 ($F34A). Ab hier beginnt eine Routine, die beim CLOSE-Befehl zuerst prüft, ob die Datei-Nummer in der Tabelle der eröffneten Datei enthalten ist. Dann holt sie die dazugehörige Geräte-Nummer und Sekundär-Adresse und schließt den Kanal und die Datei.

Adresse 798 und 799 ($31E und $31F)

Vektor auf die CHKIN-Routine des Betriebssystems

Diese Routine beginnt ab Adresse 61966 ($F20E) – beim VC 20 ab 62151 ($F2C7). Sie eröffnet einen Datenkanal zur Übernahme von Daten von dem Gerät, das durch den OPEN-Befehl angegeben worden ist.

Adresse 800 und 801 ($320 und $321)

Vektor auf die CKOUT-Routine des Betriebssystems

Dieser Vektor zeigt auf die Adresse 62032 ($F250) – beim VC 20 auf 62217 ($F309). Dort beginnt die Routine, welche einen Datenkanal zur Abgabe von Daten an das im OPEN-Befehl angegebene Gerät aufmacht.

Adresse 802 und 803 ($322 und $323)

Vektor auf die CLRCHN-Routine des Betriebssystems

Der Name dieser Routine ist die Abkürzung für »clear channel«. Diese Routine, die ab Adresse 62259 ($F333) – beim VC 20 ab 62451 ($F3F3) – beginnt, setzt alle Kanäle in den Einschaltzustand zurück. Das heißt, das Eingabegerät ist die Tastatur, das Ausgabegerät ist der Bildschirm.

Adresse 804 und 805 ($324 und $325)

Vektor auf die CHRIN-Routine des Betriebssystems

Dieser Vektor zeigt auf die Adresse 61783 ($F157) – beim VC 20 auf 61966 ($F20E). Die hier beginnende Routine, deren Abkürzung »Character Input« bedeutet, holt das jeweils nächste Byte vom Eingabepuffer des angewählten Gerätes, sofern ein solcher eingerichtet ist (zum Beispiel Kassettenpuffer, RS232-Puffer).

Bei Eingabe von der Tastatur holt diese Routine so lange Bytes aus dem Tastaturpuffer und zeigt sie auf dem Bildschirm an, bis das Zeichen für ein ungeSHIFTetes RETURN auftritt. Erst dann gibt die Routine das erste Zeichen der logischen Zeile auf dem Bildschirm an den Basic-Übersetzer weiter.

Adresse 806 und 807 ($326 und $327)

Vektor auf die CHROUT-Routine des Betriebssystems

Die CHROUT-Routine entspricht der CHRIN-Routine in der anderen Richtung. Sie bedeutet »Character Output« und transferiert ein Byte, das im Akkumulator steht, in den Puffer des angewählten Ausgabegerätes. Sie beginnt ab Adresse 62898 ($F1CA), – beim VC 20 ab 62074 ($F27A).

Adresse 808 und 809 ($328 und $329)

Vektor auf die STOP-Routine des Betriebssystems

Der Vektor zeigt auf die Adresse 63213 ($F6ED) – beim VC 20 auf 63344 ($F770). Die dort beginnende Routine prüft, ob die STOP-Taste gedrückt ist. Durch Verbiegen dieses Vektors kann die STOP-Taste abgeschaltet werden. Beim C 64 geht dies mit `POKE 808,239`; wieder eingeschaltet wird die STOP-Taste mit `POKE 808,237`. Beim VC 20 sind die Werte `POKE 808,100` beziehungsweise `POKE 808,112`.

Adresse 810 und 811 ($32A und $32B)

Vektor auf die GETIN-Routine des Betriebssystems

Diese Routine ist fast identisch mit der CHRIN-Routine (siehe Speicherzellen 804 bis 805). Sie holt genauso Zeichen von angewählten Geräten in die Eingabepuffer. Der einzige und damit wichtigste Unterschied liegt in der Behandlung der Tastatur-Eingabe. Im Gegensatz zu CHRIN holt sie ein Byte aus dem Tastaturpuffer sofort in den Akkumulator. Der Vektor zeigt auf den Anfang der Routine ab Speicherzelle 61785 ($F13E) – beim VC 20 ab 61941 ($F1F5).

Adresse 812 und 813 ($32C und $32D)

Vektor auf die CLALL-Routine des Betriebssystems

CLALL ist die Abkürzung für Close ALL (Channels and Files). Diese Routine, die ab Adresse 62255 ($F32F) – beim VC 20 ab 62447 ($F3EF) – beginnt, setzt die Speicherzelle 152 auf 0 und schließt so zwangsläufig alle Dateien und Kanäle.

Adresse 814 und 815 ($32E und $32F)

Freier Vektor

Nach dem Einschalten zeigt dieser Vektor auf die BREAK-Routine, genauso wie der Vektor in Speicherzelle 790 und 791. Er ist ein Überbleibsel aus dem PET-Betriebssystem, das aber beim VC 20 und C 64 keine Rolle spielt. Hier können also eigene Vektoren definiert und eingesetzt werden.

Adresse 816 und 817 ($330 und $331)

Vektor auf die LOAD-Routine des Betriebssystems

Dieser Vektor zeigt auf die Adresse 62622 ($F49E) – beim VC 20 auf 62793 ($F549). Die dort beginnende Routine transferiert Daten von einem Eingabegerät direkt in den RAM-Speicher. Sie kann auch zum VERIFYen durch Vergleich der geladen mit den gespeicherten Daten verwendet werden.

Adresse 818 und 819 ($332 und $333)

Vektor auf die SAVE-Routine des Betriebssystems

Diese Routine ist das Gegenstück zur LOAD-Routine. Sie beginnt ab Adresse 62941 ($F5DD) – beim VC 20 ab 63109 ($F685).

Adresse 820 bis 827 ($334 bis $33B)

Freier Speicherbereich

Diese 8 Byte stehen zur freien Verfügung.

Adresse 828 bis 1019 ($33C bis $3FB)

Kassettenpuffer

Diese 192 Byte beherbergen den Kassettenpuffer. Der Name kennzeichnet diesen Speicherbereich als Zwischenspeicher für Ein- und Ausgabe-Operationen von und auf Band.

Dabei unterscheiden sich die normalen LOAD-, SAVE- und VERIFY-Befehle von den Datei-Befehlen INPUT#, GET# und PRINT#.

Bei LOAD, SAVE und VERIFY steht im Kassettenpuffer lediglich der Vorspann, der auf englisch »Tape Header« heißt. Die Funktion und Zusammensetzung des Tape Headers habe ich schon bei den Speicherzellen 183 bis 187 und im Texteinschub Nr. 20 »Tape Header« detailliert beschrieben. Die eigentlichen Daten berühren den Kassettenpuffer nicht, sondern werden direkt von und in den RAM-Speicher transferiert.

Bei GET#, INPUT# und PRINT# werden nicht nur der Tape Header, sondern auch alle Daten im Kassettenpuffer zwischengespeichert. Dieser blockweise Transport ist an den charakteristischen Unterbrechungen des Datasettenmotors leicht zu erkennen.

Der Kassettenpuffer kann durch Verbiegen der Zeiger in Speicherzelle 178 und 179 auf beliebige Plätze des Speichers, aber nicht unterhalb 512, geschoben werden. Normalerweise gibt das keinen Sinn, es sei denn, der Speicherbereich 828 bis 1019 wurde mit einem eigenen Maschinenprogramm belegt, und durch das Verschieben des Kassettenpuffers in höhere Regionen möchte man das Maschinenprogramm vor der Zerstörung durch ungeplante Datasetten-Operationen schützen.

Die Kenntnis der Inhalte der Speicherzellen des Kassettenpuffers kann man ausnutzen, um die ärgerlichen LOAD ERROR-Probleme zu lösen. Die Methode dazu ist im Texteinschub Nr. 36 »Reparatur von LOAD ERROR« beschrieben.

Ist die Datasette nicht angeschlossen, oder wird sie nicht eingesetzt, kann der Speicherbereich des Kassettenpuffers als freier Speicher benutzt werden.

Adresse 1020 bis 1023 ($3FC bis $3FF)

Freie Speicherplätze

Auch diese 4 Byte stehen zur freien Verfügung.

Liebe Leser, wir sind am Ziel unserer Wanderung durch die Speicherlandschaft des C 64 beziehungsweise des VC 20 angelangt.

Einzelne Beschreibungen der folgenden Speicherzellen besonders der verschiedenen Register, sind ja schon im 64'er veröffentlicht worden.

Detaillierte »Wandervorschläge« müßte ich erst zusammenstellen und ausprobieren. Deshalb mache ich bei 1023 Schluß. Ich hoffe, dieser Kurs bietet Ihnen ein kleines Nachschlagewerk und regt Sie mit den Texteinschüben und Kochrezepten zum Experimentieren an. Gerade das ist die beste Methode, Ihren Computer besser kennenzulernen.

(Dr. H. Hauck/ah)

Texteinschub Nr. 35
Dem Computer ins Wort fallen

Jedesmal, wenn ein Computer eingeschaltet wird, würden seine vielen Schaltkreise und Speicherzellen irgendwelche ungeordneten Zahlen enthalten, wenn nicht ein bestimmter Schaltkreis ein RESET-Signal erzeugte. Dieses spezielle Signal geht an alle wichtigen Teile des Computers, nämlich an den Mikroprozessor und an die Bausteine für Ein- und Ausgabe.

Dadurch wird der Computer in einen definierten Anfangszustand versetzt, in dem entweder das Betriebssystem oder, falls vorhanden, ein selbststartendes Steckmodul die Befehlsgewalt erhält.

Die fest vorgegebenen Programmschritte dieser beiden lassen jedoch ein Arbeiten mit dem Computer ohne weiteres nicht zu. Wir könnten nämlich kein Resultat an ein Ausgabegerät (Drucker, Floppy, Datasette, Bildschirm) geben, und wir könnten auch keine Daten eingeben (Tastatur, Floppy, Datasette).

Der Computer wäre nicht steuerbar, wenn wir ihn nicht in seinem vorgegebenen Programmablauf unterbrechen könnten.

Die Unterbrechungsmöglichkeit heißt in der Fachsprache »INTERRUPT«.

Im Gegensatz zu den Großrechenanlagen, die meistens mit vielen Klassen von Interrupts ausgerüstet sind, haben die Heim-Computer von Commodore nur zwei Arten:

- IRQ – der Interrupt Request
- NMI – der Non Maskable Interrupt

Ich habe nicht vor, Ihnen alle Details der Interrupt-Technik zu erklären. Das geht weit über den normalen Umfang meiner Texteinschübe hinaus. In anderen Aufsätzen können Sie mehr darüber erfahren, zum Beispiel von Helmut Welke in Ausgabe 11/84, Seite 84, oder im Assembler-Kurs von Heimo Ponnath in den Ausgaben 7 bis 9/85 und Sonderheft 8/85 (Assembler).

Aber einige Erklärungen, so hoffe ich jedenfalls, werden Ihnen auch hier das Interrupt-Prinzip deutlich machen.

Die beiden oben genannten Unterbrechungsarten unterscheiden sich sowohl dadurch, wer die Unterbrechung auslösen kann, als auch in der Art, wie sie gehandhabt werden.

NMI-Auslöser
sind Signale der RS232-Schnittstelle und der Autostart-Steckmodule. Dazu kommen noch die RESTORE-Taste, wenn sie gleichzeitig mit der RUN/STOP-Taste gedrückt wird und der CIA #2 beziehungsweise der VIA #1.

Wie gesagt, nähere Einzelheiten darüber finden Sie in den oben genannten Aufsätzen.

IRQ-Auslöser
ist 60mal in der Sekunde das Betriebssystem selbst, um die Werte von TI und TI$ höher zu setzen, um zu prüfen, ob die STOP-Taste gedrückt ist, um das Cursorblinken zu erzeugen, um die Tasten der Datasette und schließlich auch die Tastatur abzufragen. Ein IRQ-Interrupt kann aber auch durch Lesen oder Schreiben vom – beziehungsweise auf das – Band, durch die serielle Schnittstelle und durch die Rasterzeilen-Abtastung ausgelöst werden. Programmierbare IRQ-Interrupts sind möglich durch Sprite-Kollisionen, durch Lichtgriffel-Signale und durch den CIA #1 beziehungsweise den VIA #2. Besonders durch die letzteren Ein-/Ausgabe-Bausteine unterscheiden sich die Interrupts von C 64 und VC 20.

NMI-Abläufe
sind schon durch ihren Namen gekennzeichnet. »Non-Maskable« heißt soviel wie »nicht unterdrückbar«. Immer, wenn ein NMI-Signal ankommt, merkt sich der Computer, was er gerade macht, unterbindet alle IRQ-Signale und springt auf eine NMI-Routine, deren Beginn mit dem Vektor in Speicherzelle 792 und 793 vorgegeben ist.

Herr Ponnath hat im Assembler-Kurs dies sehr treffend mit dem überkochenden Kessel auf dem Herd verglichen, der heruntergestellt werden muß, selbst wenn gerade die Türglocke klingelt, was uns normalerweise beim Lesen der Zeitung unterbrechen würde.

Erst in der NMI-Routine werden nach einer vorgegebenen Prioritätsliste alle NMI-Auslöser der Reihe nach abgefragt, bis der Verursacher gefunden ist.

IRQ-Abläufe
sind maskierbar, das heißt sie können, wie gerade gesagt, unterdrückt werden, entweder durch programmiertes Abschalten – das entspricht dem Abstellen der Türglocke – oder durch ein NMI-Signal.

Bei einem IRQ-Signal wird zuerst der gerade laufende Befehl noch bearbeitet, dann startet die IRQ-Routine, deren Beginn durch den Vektor in Speicherzelle 788 und 789 vorgegeben ist. In dieser Routine wird entschieden, ob der IRQ-Interrupt durch den Maschinencode-Befehl BRK (Break) oder durch angeschlossene Peripheriegeräte ausgelöst worden ist.

Wir sehen also, daß die Unterbrechungen einer festgelegten Priorität unterworfen sind. Ihre Steuerung aber erfolgt immer so, daß keine Interrupt-Anmeldung verlorengeht, sondern jede in der gebührenden Reihenfolge abgearbeitet wird.

Schließlich sei noch hervorgehoben, daß der Sprung in die Interrupt-Routinen über die Vektoren die Möglichkeit eröffnet, diese Routinen nach eigenem Geschmack abzuändern beziehungsweise durch eigene Routinen zu ersetzen.

Texteinschub Nr. 36
Reparatur eines LOAD ERRORs

Die Datasette – das Bandgerät von Commodore – ist sicher eines der sichersten und zuverlässigsten seiner Art.

Und doch weigert sie sich gelegentlich, ein Programm vom Band in den Computer zu laden. Alles, was der Computerfreund erhält, ist die Fehlermeldung LOAD ERROR auf dem Bildschirm.

Natürlich: die nächstliegende Maßnahme ist, den LOAD-Vorgang zu wiederholen. Bringt auch das keinen Erfolg, muß die Flinte noch lange nicht ins Korn geworfen werden. Eine kleine Diagnose und die Kenntnis des Tape Headers im Kassettenspeicher (Speicherzelle 828 bis 1023) kann in den meisten Fällen weiterhelfen.

Die 1. Diagnose:
Wenn ein Programm auf Band geSAVEt wird, tut das der C 64 und VC 20 zur Sicherheit gleich zweimal, mit zwei völlig identischen Blöcken. Beim Laden des Programms wird der erste Block in den Arbeitsspeicher des Computers geladen.

Anschließend wird Byte für Byte der zweite Block vom Band mit dem ersten Block im Speicher verglichen. Übersteigt die Anzahl der dabei gefundenen Fehler ein bestimmtes Maß, dann bricht der Computer mit LOAD ERROR ab.

!! Der erste Programmblock steht aber immer noch im Arbeitsspeicher !!

Um zu sehen, ob er in Ordnung oder halbwegs brauchbar ist, machen Sie bitte nach der Fehlermeldung gar nichts – kein RUN, kein RESTORE – sondern LISTen Sie lediglich das Programm. Besteht es nur aus verfälschten Zeilen und Symbolen, dann ist nicht mehr viel zu retten.

Ist es aber fast oder völlig intakt, können wir es retten. Doch auch jetzt ist noch Vorsicht geboten. Lassen Sie das Programm in Ruhe und heben Sie sich die Korrekturen etwaiger Fehler für später auf.

Die 2. Diagnose:
Sie betrifft den Tape Header. Vor dem Laden des ersten Programmblocks in den Arbeitsspeicher kommt der Tape Header in den Kassettenpuffer (siehe den Texteinschub Nr. 20 »Tape Header«).

In Speicherzelle 828 steht ein Kennzeichen-Byte, in 829 und 830 in Low-/High-Byte-Darstellung die Adresse, ab der das Programm im Arbeitsspeicher steht.

Für uns ist aber die Adresse wichtig, die in Speicherzelle 831 und 832 steht. Sie nennt dem Betriebssystem nämlich die Endadresse des Programms im Arbeitsspeicher. Diese Adresse wird nach dem erfolgreichen Abschluß des Ladevorgangs in die Speicherzellen 45 und 46, 47 und 48, 49 und 50 eingeschrieben.

Ich sagte: »nach dem erfolgreichen Ladevorgang«. Und das gerade ist ja leider nicht eingetreten – deswegen können wir den akzeptablen ersten Programmblock im Arbeitsspeicher nicht RUNen, korrigieren und sonstwie verarzten, nur LISTen.

Reparatur:

Da durch den Abbruch die Zeiger in oben genannten drei Speicherzellenpaaren nicht gesetzt worden sind, holen wir das ganz einfach manuell nach mit der folgenden Direkteingabe:

```
POKE 45,PEEK(831): POKE 46,PEEK(832):
POKE 47,PEEK(831): POKE 48,PEEK(832):
POKE 49,PEEK(831): POKE 50,PEEK(832):
```

Das geht auch etwas eleganter und kürzer:

```
FOR K=45 TO 49 STEP 2: POKE K,PEEK(831): POKE K+1,PEEK(832):
NEXT
```

Damit sind die Zeiger richtig gesetzt, und Sie haben Ihr Programm wieder. Erst jetzt dürfen Sie eventuelle Fehler korrigieren.

Ich habe nicht erwähnt, was die Zeiger in 45 und 46, 47 und 48, 49 und 50 bedeuten. Aber das steht ja schließlich in der Memory Map.

```
6000 FOR A=828 TO 853
6010 READ B
6020 POKE A,B
6030 NEXT:END
6040 DATA 120,169,73,141,20,3,169,3,
6050 DATA 141,21,3,88,96,169,1,141,139
6060 DATA 2,169,0,141,140,2,76,49,234
```

Listing 1. Programm zur Änderung der Tastenwiederholgeschwindigkeit

```
828  SEI        setzt die Interrupt Enable Flagge
829  LDA #73    lädt Akku mit der Zahl 73
831  STA 788    schreibt die 73 in Zelle 788
834  LDA #3     lädt Akku mit der Zahl 3
836  STA 789    schreibt die 3 in die Zelle 789
839  CLI        löscht die Interrupt Enable Flagge
840  RTS        Ende des Unterprogramms
841  LDA #1     lädt Akku mit der Zahl 1
843  STA 651    schreibt die 1 in Zelle 651
846  LDA #0     lädt Akku mit der 0
848  STA 652    schreibt die 0 in die Zelle 652
851  JMP 59953  Sprung auf Speicherzelle 59953 zum  Weiterlauf
der normalen Interrupt-Routine
```

Listing 2. Disassembliertes Programm

```
,033C  78        SEI
,033D  A9 49     LDA #49
,033F  8D 14 03  STA 0314
,0342  A9 03     LDA #03
,0344  8D 15 03  STA 0315
,0347  58        CLI
,0348  60        RTS
,0349  A9 01     LDA #01
,034B  8D 8B 02  STA 028B
,034E  A9 00     LDA #00
,0350  8D 8C 02  STA 028C
,0353  4C 31 EA  JMP EA31
```

Listing 3. Disassembliertes Programm mit Hexdump

Inhaltsverzeichnis der Texteinschübe und Tabellen

Die Bedeutung der Speicherzellen 0 bis 1023, nach Funktionen geordnet

Bandoperationen

146	\$92	Zeitkonstante beim Lesen vom Band
147	\$93	Flagge für LOAD oder VERIFY
150	\$96	Arbeitsspeicher für Band-Leseroutinen
153	\$99	Nummer des Eingabegerätes
155	\$9B	Fehlerkontrolle bei Bandoperationen
156	\$9C	Flagge für korrektes Byte vom Band
158-159	\$9E-\$9F	Zwischenspeicher bei Kassettenoperationen

Dezimal	Hex	Bedeutung
165	$A5	Zähler für Band-Synchronisierung
167	$A7	Zwischenspeicher für Kassettenroutinen
168	$A8	Bitzähler bei Band-Ein-/Ausgabe
170	$AA	Zwischenspeicher für Kassettenroutinen
171	$AB	Quersummenprüfung und Zähler für Band-Header
172-173	$AC-$AD	Zeiger auf die Anfangsadresse für Ein-/Ausgabe
174-175	$AE-$AF	Zeiger auf die Endadresse für Ein-/Ausgabe
176-177	$B0-$B1	Zeitkonstante beim Lesen vom Band
178-179	$B2-$B3	Zeiger auf den Kassettenpuffer
181	$B5	Blockangabe bei Kassettenoperationen
182	$B6	Ausgabe-Zwischenspeicher
183	$B7	Länge des File-Namens
185	$B9	Sekundär-Adresse
186	$BA	Geräte-Nummer
187-188	$BB-$BC	Zeiger auf Adresse des derzeitigen File-Namens
189	$BD	Zwischenspeicher für Zeichen
190	$BE	Blockzähler für Kassetten-Ein-/Ausgabe
191	$BF	Zwischenspeicher für LOAD-Operationen vom Band
192	$CO	Sperre des Motors der Datasette
193-194	$C1-$C2	Anfangsadresse für Ein-/Ausgabe-Operationen
195-196	$C3-$C4	Zeiger auf den Anfang des Programms hinter dem Tape Header
256-318	$100-$13E	Arbeitsspeicher für Fehler bei der Eingabe vom Band
Bildschirm-Cursor		
9	$9	Spaltenposition des Cursors vor dem letzten TAB- oder SPC-Befehl
200	$C8	Zeiger auf das Ende der eingegebenen logischen Zeile
201-202	$C9-$CA	Zeiger auf Zeilen- und Spaltenposition des letzten Zeichens einer Zeile
204	$CC	Schalter für Cursorblinken
205	$CD	Zähler für Blinkfrequenz des Cursors
206	$CE	Bildschirmcode des Zeichens unter dem Cursor
207	$CF	Flagge für Blinkzustand des Cursors
209-210	$D1-$D2	Zeiger auf den Anfang der Bildschirmzeile, auf welcher der Cursor gerade steht
211	$D3	Position des Cursors innerhalb einer logischen Zeile
214	$D6	Nummer der echten Zeile, in der sich der Cursor gerade befindet
647	$287	Zeichenfarbe unter dem Cursor
Bildschirm-Farbe		
243-244	$F3-$F4	Position des Cursors im Farbspeicher
646	$286	Aktuelle Farbe der Zeichen (Vordergrundfarbe)
647	$287	Zeichenfarbe unter dem Cursor
Bildschirm-Zeichen		
199	$C7	Flagge für reverse Darstellung der Zeichen
206	$CE	Bildschirmcode des Zeichens unter dem Cursor
212	$D4	Flagge für Gänsefuß-Modus
215	$D7	Zwischenspeicher für den ASCII-Codewert der zuletzt gedrückten Taste
216	$D8	Flagge für Insert-Modus
Bildschirm-Zeilen		
200	$C8	Zeiger auf das Ende der eingegebenen logischen Zeile
201-202	$C9-$CA	Zeiger auf Zeilen- und Spaltenposition des letzten Zeichens einer Zeile
209-210	$D1-$D2	Zeiger auf den Anfang der Bildschirmzeile, auf welcher der Cursor gerade steht
211	$D3	Position des Cursors innerhalb einer logischen Zeile
213	$D5	Länge einer Bildschirmzeile
214	$D6	Nummer der echten Zeile, auf der sich der Cursor gerade befindet
217-242	$D9-$F2	Link-Tabellen der Bildschirm-Zeilen
658	$292	Flagge für Scrollen
DATA		(siehe READ)
Datei		
152	$98	Anzahl der offenen Dateien
153	$99	Nummer des Eingabegerätes
154	$9A	Nummer des Ausgabegerätes
183	$B7	Länge des derzeitigen Datei-Namens
184	$B8	Nummer der derzeitigen Datei
185	$B9	Derzeitige Sekundär-Adresse
186	$BA	Derzeitige Geräte-Nummer
187-188	$BB-$BC	Zeiger auf Adresse des derzeitigen Datei-Namens
601-610	$259-$262	Tabelle der Datei-Nummern
611-620	$263-$26C	Tabelle der Geräte-Nummern
621-630	$26D-$276	Tabelle der Sekundär-Adressen
DIM		
11	$B	Anzahl der Dimensionen von Feldern (Arrays)
12	$C	Flagge für Basic-Routinen, die ein Feld suchen, beziehungsweise aufbauen
Eingabe-Puffer		
7	$7	Suchzeichen zur Prüfung von Basic-Texteingabe
8	$8	Suchzeichen speziell für Befehlsende und Gänsefüße
11	$B	Flagge für den Eingabe-Puffer
512-600	$200-$258	Eingabe-Puffer von Basic
Einschalten/Reset (beeinflußte Adressen)		
0-2	$0-$2	Sprungbefehl und wählbare Sprungadresse beim USR-Befehl (nur VC 20)
3-4	$3-$4	Vektor auf die Routine zur Umwandlung einer Gleitkommazahl in eine ganze Zahl mit Vorzeichen
5-6	$5-$6	Vektor auf die Routine zur Umwandlung einer ganzen Zahl in eine Gleitkommazahl
19	$13	Flagge zur Kennzeichnung des laufenden Ein-/Ausgabegerätes
22	$16	Zeiger auf freien Speicherplatz im String Descriptor Stack
43-44	$2B-$2C	Zeiger auf Anfang der Basic-Programme im Speicher
45-46	$2D-$2E	Zeiger auf Anfang der Variablen im Speicher (nur bei Reset)
51-52	$33-$34	Zeiger auf die untere Grenze des Speicherbereichs für den Text der Zeichenketten-Variablen
55-56	$37-$38	Zeiger auf das Ende des für Basic-Programme verfügbaren Speichers
122-123	$7A-$7B	Teil der CHRGET-Routine
139-143	$8B-$8F	Wert der RND-Funktion als Gleitkommazahl
153	$99	Nummer des Eingabe-Gerätes
154	$9A	Nummer des Ausgabe-Gerätes
160-162	$A0-$A2	Interne Uhr für TI und TI$ (nur beim Einschalten)
178-179	$B2-$B3	Zeiger auf den Kassetten-Puffer
195-196	$C3-$C4	Zeiger auf den Anfang des Programms hinter dem Tape Header
256-511	$100-$1FF	Stapelspeicher (Stack)
641-642	$281-$282	Zeiger auf den Anfang des Programmspeichers
643-644	$283-$284	Zeiger auf das Ende des Programmspeichers
646	$286	Aktuelle Farbe der Zeichen (Vordergrundfarbe)
648	$288	Beginn des Bildschirmspeichers
655-656	$28F-$290	Vektor auf die Routine der Tastencode-Tabellen
784-786	$310-$312	nur C64, identisch mit 0-3 beim VC 20
788-819	$314-$333	Indirekte Sprungvektoren auf Routinen des Betriebssystems
END		
57-58	$39-$3A	Nummer der laufenden Basic-Programmzeile
59-60	$3B-$3C	Zeilennummer der letzten Programmunterbrechung
61-62	$3D-$3E	Zeiger auf die Adresse, ab welcher der Text der laufenden Basic-Zeile abgespeichert ist
Felder (Arrays)		
11	$B	Anzahl der Dimensionen von Feldern (Arrays)
12	$C	Flagge für Basic-Routinen, die ein Feld suchen, beziehungsweise aufbauen
16	$10	Flagge zur Anzeige eines Variablenfeldes oder einer selbstdefinierten Funktion
47-48	$2F-$30	Zeiger auf die Anfangsadresse des Speicherbereiches für Felder (Arrays)
49-50	$31-$32	Zeiger auf die Endadresse des Speicherbereichs für Felder (Arrays)
FN		
16	$10	Flagge zur Anzeige eines Variablenfeldes oder einer selbstdefinierten Funktion
78-79	$4E-$4F	Zeiger auf Adresse, ab welcher der Wert der Variablen einer selbst definierten Funktion gespeichert ist
FOR-NEXT		
47-48	$2F-$30	Zeiger auf die Anfangsadresse des Speicherbereichs für Felder (Arrays)
57-58	$39-$3A	Nummer der laufenden Basic-Programmzeile
73-74	$49-$4A	Zwischenspeicher für Variable einer FOR-NEXT-Schleife
FRE		
49-50	$31-$32	Zeiger auf die Endadresse des Speicherbereichs für Felder (Arrays)
51-52	$33-$34	Zeiger auf die untere Grenze des Speicherbereichs für den Text der Zeichenketten-Variablen
Garbage Collection		
15	$F	Flagge bei LIST, Garbage Collection und Textumwandlung
49-50	$31-$32	Zeiger auf die Endadresse des Speicherbereichs für Felder (Arrays)
51-52	$33-$34	Zeiger auf die untere Grenze des Speicherbereichs für den Text der Zeichenketten-Variablen
83	$53	Flagge für Garbage Collection
GET		
17	$11	Flagge für INPUT, GET oder READ
18	$13	Flagge zur Kennzeichnung des laufenden Ein-/Ausgabegerätes
67-68	$43-$44	Zeiger auf die Adresse, aus welcher die Befehle INPUT, GET und READ die Zeichen/Zahlen holen
GET#		
19	$13	Flagge zur Kennzeichnung des laufenden Ein-/Ausgabegerätes
153	$99	Nummer des Eingabegerätes
Gleitkomma		
3-4	$3-$4	Vektor auf die Routine zur Umwandlung einer Gleitkommazahl in eine ganze Zahl mit Vorzeichen
5-6	$5-$6	Vektor auf die Routine zur Umwandlung einer ganzen Zahl in eine Gleitkommazahl



Dezimal	Hex	Bedeutung
97–102	$61–$66	Gleitkomma-Akkumulator Nr. 1
104	$68	Überlauf-Speicher des Gleitkomma-Akkumulators Nr.1
105–110	$69–$6E	Gleitkomma-Akkumulator Nr. 2
111	$6F	Flagge für Vorzeichenvergleich der Gleitkomma-Akkumulatoren 1 und 2
112	$70	Rundungsspeicher des Gleitkomma-Akkumulators Nr.1
255	$FF	Zwischenspeicher von Gleitkommazahlen in ASCII-Werte
256–266	$100–$10A	Arbeitsspeicher für Umwandlung von Gleitkommazahlen in ASCII-Werte
778–779	$30A–$30B	Indirekter Sprungvektor auf die Basic-Routine, die einen numerischen Ausdruck in eine Gleitkommazahl umwandelt
GOTO		
20–21	$14–$15	Zeilennummer für LIST, GOTO, GOSUB und ON
57–58	$39–$3A	Nummer der laufenden Basic-Programmzeile
INPUT		
17	$11	Flagge für INPUT, GET oder READ
67–68	$43–$44	Zeiger auf die Adresse, aus welcher die Befehle INPUT, GET und READ die Zeichen/Zahlen holen
INPUT#		
19	$13	Flagge zur Kennzeichnung des laufenden Ein-/Ausgabegerätes
153	$99	Nummer des Eingabegerätes
INST		
212	$D4	Flagge für Gänsefuß-Modus
216	$D8	Flagge für INSERT-Modus
Interrupt mit BREAK		
57–58	$39–$3A	Nummer der laufenden Basic-Programmzeile
170	$AA	Zwischenspeicher für Kassettenroutinen
663	$297	RS232-Status-Register
790–791	$316–$317	Vektor auf die BREAK-Interrupt-Routine
Interrupt mit IRQ		
671–672	$29F–$2A0	Zwischenspeicher für den IRQ-Vektor während Kassetten-Ein-/Ausgabe
788–789	$314–$315	Vektor auf die IRQ-Interrupt-Routine
Interrupt mit NMI		
792–793	$318–$319	Vektor auf die NMI-Interrupt-Routine
Kassettenpuffer		
166	$A6	Zähler der bearbeiteten Bytes im Kassettenpuffer
178–179	$B2–$B3	Zeiger auf den Kassettenpuffer
828–1019	$33C–$3FB	Kassettenpuffer
LIST		
15	$F	Flagge bei LIST, Garbage Collection und Text-umwandlung
20–21	$14–$15	Zeilennummer für LIST, GOTO, GOSUB und ON
LOAD/VERIFY		
10	$A	Flagge für LOAD oder VERIFY
147	$93	Flagge für LOAD oder VERIFY
172–173	$AC–$AD	Zeiger auf die Anfangsadresse für Ein-/Ausgabe
174–175	$AE–$AF	Zeiger auf die Endadresse für Ein-/Ausgabe
183	$B7	Länge des File-Namens
185	$B9	Sekundär-Adresse
187–188	$BB–$BC	Zeiger auf Adresse des derzeitigen File-Namens
195–196	$C3–$C4	Zeiger auf den Anfang des Programms hinter dem Tape Header
816–817	$330–$331	Indirekter Sprungvektor auf die LOAD-Routine des Betriebssystems
NEXT (siehe FOR)		
READ DATA		
17	$11	Flagge für INPUT, GET oder READ
63–64	$3F–$40	Zeilennummer des gerade laufenden DATA-Befehls
65–66	$41–$42	Zeiger auf die Adresse, ab der die laufenden DATA-Angaben gespeichert sind
67–68	$43–$44	Zeiger auf die Adresse, aus welcher die Befehle INPUT, GET und READ die Zeichen/Zahlen holen
75–76	$4B–$4C	Zwischenspeicher für Zeiger bei READ und bei mathematischen Operationen
RESET (siehe Einschalten)		
RND		
139–143	$8B–$8F	Wert der RND-Funktion als Gleitkommazahl
RS232-Schnittstelle		
167	$A7	Zwischenspeicher für Eingabe über die RS232-Schnittstelle
168	$A8	Bitzähler für RS232-Eingabe
169	$A9	RS232-Flagge für Startbit-Prüfung
170	$AA	RS232-Eingabespeicher
171	$AB	Paritygrüfung
181	$B5	RS232-Anzeige für nächstes Bit
182	$B6	Ausgabe-Zwischenspeicher für RS232
189	$BD	Zwischenspeicher für RS232-Parity-Prüfung
247–248	$F7–$F8	Zeiger auf den Anfang des RS232-Eingabepuffers
249–250	$F9–$FA	Zeiger auf den Anfang des RS232-Ausgabepuffers
659	$293	RS232-Steuerregister
660	$294	RS232-Befehlsregister
661–662	$295–$296	RS232 frei wählbare Baudrate

Dezimal	Hex	Bedeutung
663	$297	RS232-Statusregister
664	$298	RS232-Anzahl der zu übertragenden Bits
665–666	$299–$29A	Zeit, die bei RS232 zum Übertragen eines Bits gebraucht wird
667	$29B	Index auf das Ende des RS232-Eingabepuffers
668	$29C	Index auf den Anfang des RS232-Eingabepuffers
669	$29D	Index auf den Anfang des RS232-Ausgabepuffers
670	$29E	Index auf das Ende des RS232-Ausgabepuffers
SAVE		
172–173	$AC–$AD	Zeiger auf die Anfangsadresse für Ein-/Ausgabe
174–175	$AE–$AF	Zeiger auf die Endadresse für Ein-/Ausgabe
818–819	$332–$333	Indirekter Sprungvektor auf die SAVE-Routine des Betriebssystems
Serielle Schnittstelle		
148	$94	Flagge für Floppy/Drucker-Ausgabe
149	$94	Zeichen im Ausgabepuffer
163–164	$A3–$A4	Zwischenspeicher
172–173	$AC–$AD	Zeiger auf die Anfangsadresse für Ein-/Ausgabe
174–175	$AE–$AF	Zeiger auf die Endadresse für Ein-/Ausgabe
193–194	$C1–$C2	Anfangsadresse für Ein-/Ausgabe-Operationen
SIN		
18	$12	Flagge für Vorzeichen des Ergebnisses bei SIN und TAN
Speicherbelegung		
43–44	$2B–$2C	Zeiger auf den Anfang der Basic-Programme im Speicher
45–46	$2D–$2E	Zeiger auf die Anfangsadresse des Speicherbereichs für Variable
47–48	$2F–$30	Zeiger auf die Anfangsadresse des Speicherbereichs für Felder (Arrays)
49–50	$31–$32	Zeiger auf die Endadresse +1 des Speicherbereichs für Felder (Arrays)
51–52	$33–$34	Zeiger auf die untere Grenze des Speicherbereichs für den Text der String-Variablen
53–54	$35–$36	Zeiger auf die Adresse des zuletzt eingegebenen Strings
55-56	$37–$38	Zeiger auf das Ende des für Basic-Programme verfügbaren Speichers
641–642	$281–$282	Zeiger auf den Anfang des Programmspeichers
643–644	$283–$284	Zeiger auf das Ende des Programmspeichers
648	$288	Beginn des Bildschirmspeichers
Speicher zur freien Verfügung		
146–150	$92–$96	nur wenn Datasette nicht benutzt wird
163–177	$A3–$B1	nur wenn Datasette oder RS232-Schnittstelle nicht benutzt wird
247–250	$F7–$FA	nur wenn RS232-Schnittstelle nicht benutzt wird
251–254	$FB–$FE	
659–670	$293–$29E	nur wenn RS232-Schnittstelle nicht benutzt wird
671–672	$29F–$2A0	nur wenn Datasette nicht benutzt wird
673–678	$2A1–$2A6	nur beim VC 20
679–767	$2A7–$2FF	
784–787	$310–$313	nur beim VC 20
820–827	$334–$33B	
828–1019	$33C–$3FB	nur wenn Datasette nicht benutzt wird
1020–1023	$3FC–$3FF	
ST(atus)		
144	$90	Status-Variable ST
663	$297	RS232-Statusregister
Stapelspeicher (Stack)		
25–33	$19–$21	Stack für vorläufige Zeichenketten
319–511	$13F–$1FF	Speicherbereich des Mikroprozessor-Stapels
STOP		
57–58	$39–$3A	Nummer der laufenden Basic-Programmzeile
59–60	$3B–$3C	Zeilennummer der letzten Programmunterbrechung
145	$91	Zwischenspeicher für Abfrage der STOP-Taste
808–809	$328–$329	Indirekter Sprungvektor auf die STOP-Routine des Betriebssystems
STRINGS		
22	$16	Zeiger auf freien Speicherplatz im String Descriptor Stack
23–24	$17–$18	Zeiger auf die Adresse der letzten Zeichenkette im Temporary String Stack
25–33	$19–$21	Descriptor Stack für vorläufige Zeichenketten
51–52	$33–$34	Zeiger auf die untere Grenze des Speicherbereichs für den Text der String-Variablen
53–54	$35–$36	Zeiger auf die Adresse des zuletzt eingegebenen Strings
80–82	$50–$52	Zeiger auf einen vorläufigen Speicherplatz einer Zeichenkette, die gerade bearbeitet wird
SYS		
780	$30C	Speicher für den Akkumulator
781	$30D	Speicher für das X-Register
782	$30E	Speicher für das Y-Register
783	$30F	Speicher für das Status-(P)-Register

Die Zeropage-Straße des C 128

Die Zeropage ist einer der wichtigsten Bereiche in unserem Computer. Dort laufen die meisten Operationen ab und liegen viele Betriebssystem-Vektoren. Hier erhalten Sie eine Übersicht über die Funktion der ersten 256 Adressen des Computer-Speichers.

Dort, wo die viele tausend Adressen umfassende Speicherstraße unseres C 128 beginnt, gewissermaßen im ältesten Teil dieser Straße, heißt sie »Zeropage«. Die Erklärung dieses Namens liegt darin, daß man eine Anzahl von 256 Häusern (Adressen) eine Page (zu deutsch: Seite) nennt und daß es sich hier um die »nullten« 256 Adressen (in Computerkreisen fängt man immer bei Null an zu zählen) unserer Speicherstraße dreht. Das englische Wort für Null ist »Zero«, daher also Zeropage.

Jedes Haus in der Zeropage-Straße ist einmalig und man kann durchaus behaupten, daß auch nur der Ausfall eines einzigen leicht zur Desorganisation unserer ganzen Speicherstadt C 128 führen kann. Als unsere Speicherstadt gegründet wurde, hieß sie KIM 1 und war ein kleines Dorf, einfach aufgebaut und überschaubar. Von Computergeneration zu Computergeneration wurde sie komplexer und eleganter. Bald reichten die 256 Häuser der Zeropage-Straße nicht mehr aus und es kamen immer neue dazu. So wurde die Zeropage-Straße um viele hundert Adressen verlängert. Im neueren Teil nennt man sie nun allgemein »Erweiterte Zeropage«. Es kam immer wieder vor, daß einige Bewohner der Häuser, die bestimmte Aufgaben wahrzunehmen haben, umgezogen oder gar weggezogen sind. Andere – ziemlich viele – sind neu dazugekommen. Zwar verfügen die Verwaltungen unserer Speicherstadt (in den Stadtteilen Basic-Interpreter oder Betriebssystem etc.) über alle aktuellen Angaben, die sie zum reibungslosen Verkehr mit den Einrichtungen der Zeropage Straße benötigen – sonst wäre ja das Funktionieren unseres Gemeinwesens nicht gewährleistet – andererseits ist es aber wünschenswert, daß auch Ortsfremde sich der vielen Möglichkeiten dieses wichtigen Teiles unserer Stadt fachkundig bedienen können.

Orientierung

Damit haben wir auch einen Schwachpunkt unserer Computerstadt angesprochen: Fachkundig! Nur allzu leicht passiert es, daß Unkundige fehlerhaft die Einrichtungen unserer wichtigsten Straße benutzen. Die Folgen waren oft gravierend und konnten nur durch einen Neustart behoben werden! Aus diesem Grund wird hiermit der Führer durch die Zeropage und die erweiterte Zeropage vorgestellt, der die wichtigsten Adressen und ihre Benutzung erläutert.

Bevor wir aber die Zeropage-Straße beschreiben, sollten Sie auch noch wissen, wie unsere Speicherstadt aussieht und wo diese Straße überhaupt zu finden ist. Im Bild 1 sehen Sie eine Gesamtansicht der C 128-Stadt:

Darin erkennen Sie, daß die Speicherstadt aus mehreren Ebenen besteht, deren untere BANK 0 genannt wird. Darüber befindet sich BANK 1. BANK 2 bis BANK 13 sind sozusagen vorbereitet für den weiteren Ausbau unserer Stadt. In den BANKs 14 und 15 befindet sich die Verwaltung des gesamten Gemeinwesens. Alle Ebenen dieser Stadt haben einen Bereich gemeinsam, der COMMON AREA genannt wird, was etwa »Gemeinsamer Bereich« bedeutet. Gleichgültig in welcher Ebene der Stadt Sie sich gerade befinden, wenn Sie nur weit genug die Straße hinuntergehen bis zu den niedrigen Hausnummern, Sie landen unweigerlich ab Adresse $0400 (und kleiner) in dieser Common Area. Noch eine interessante Eigenschaft der Ebenen 14 und 15. Deren niedrigste Hausnummer ist nämlich $4000. Unterhalb dieser Hausnummer befinden Sie sich plötzlich wieder in BANK 0. Daher gibt es für den Neuling in unserer Stadt ein recht einfaches Rezept, die Zeropage-Straße und ihre Verlängerung zu finden: Einfach zu den Hausnummern unter $0400 gehen! Wir werden später noch erkennen, daß die Straße der erweiterten Zeropage sogar noch Hausnummern enthält, die größer als $0400 sind. Um diese zu finden, muß der Besucher dann am besten mittels des Befehls BANK 0 in die untere Etage gehen (BANK 15 und BANK 14 sind aber – wegen der eben erklärten Eigenart dieser Stadtteile – auch erlaubt).

Die beiden ersten Häuser

Noch eine technische Bemerkung: Es ist an dieser Stelle natürlich nicht möglich, wirklich jedes Haus dieser Straße mit gleicher Aufmerksamkeit vorzustellen. Einige sind in der Beschreibung sogar weggelassen worden. Einige andere enthalten lediglich den Hinweis auf einen anderen Führer, den Sie in diesem Heft vollständig abgedruckt finden: »Memory Map mit Wandervorschlägen« von Dr. H. Hauck. Jener hatte die C 64-Stadt beschrieben, aus der dann unsere Stadt entstand. Überall dort also, wo die einzelnen Häuser noch dieselbe Funktion wie damals haben, finden Sie einen Verweis auf das Werk dieses exzellenten Kenners jener Stadt.

Sie werden sich erinnern: Man fängt bei Null an zu zählen. Daher haben die beiden ersten Häuser der Zeropage-Straße die Nummern 0 und 1. Beide haben Namen und in ihnen werden bestimmte – gleich noch zu beschreibende – Arbeiten geleistet:

Adresse	Name	Aufgabe
00	D8502	8502 Datenrichtungsregister
01	R8502	8502 Datenregister

Jedes dieser Häuser (Bytes) besteht aus 8 Zimmern (Bits), deren Inhalt bedeutend ist für einige Computerfunktionen. Die Zuordnung der Aufgaben auf die Zimmer ist in beiden Adressen identisch, lediglich die Bedeutung des Inhaltes ist unterschiedlich. In Register 00 entscheidet dieser Inhalt nur darüber, ob eine Leitung auf Ausgang (dann ist das Bit = 1) oder Eingang (dann ist es 0) geschaltet wird. In Adresse 01 haben die Zimmer folgende Funktion:

Bit	Funktion
7	unbenutzt, steht auf 0
6	Zustand der ASCII/DIN-Taste

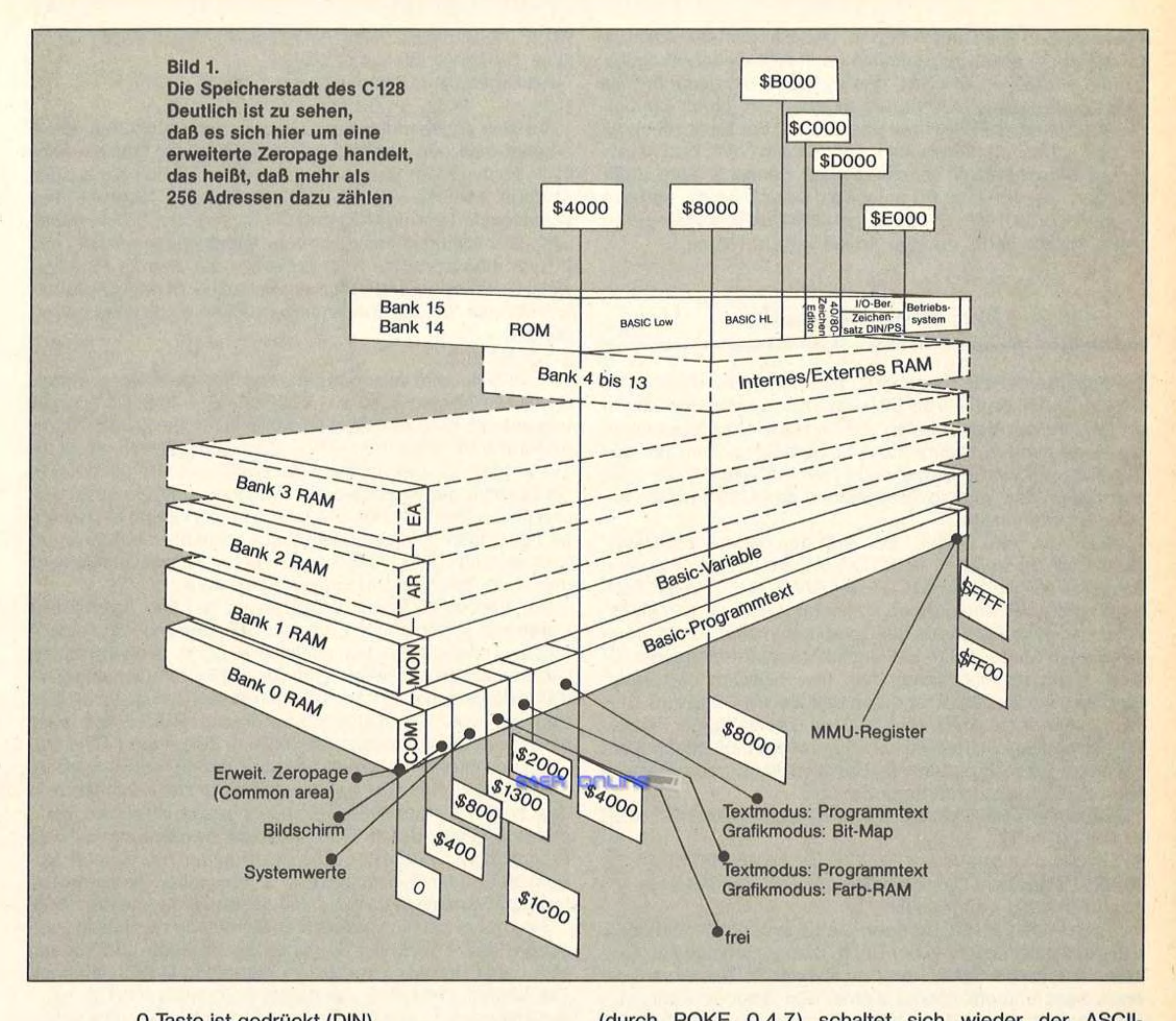


Bild 1. Die Speicherstadt des C128 Deutlich ist zu sehen, daß es sich hier um eine erweiterte Zeropage handelt, das heißt, daß mehr als 256 Adressen dazu zählen

0 Taste ist gedrückt (DIN)
1 Taste ist nicht gedrückt (ASCII)

5 Kassettenmotorsteuerung
- 0 Motor ein
- 1 Motor aus

4 Datasettentaste
- 0 Taste gedrückt
- 1 keine Taste gedrückt

3 Serielle Schreibleitung Datasette

2 und 1 Steuerung von Text- und Grafikmodus
- 01 Textmodus
- 10 Grafikmodus

0 Inhalt von $D800 bis $DC00
- 0 Farb-RAM für Grafik (Multicolorfarbe 2)
- 1 Farb-RAM für Text

Der normale Inhalt von 00 ist $2F = 47, der von 01 bei ASCII-Betrieb ist $73 = 115. Ist der DIN-Zeichensatz aktiv, dann findet man in 01 statt dessen $33 = 51. Eine Umschaltung von ASCII auf DIN ohne Benutzung der Taste und eine gleichzeitige Sperrung dieser Taste erreicht man auf folgende Weise:

POKE 0,111 schaltet die Leitung, die durch Bit 6 des Datenrichtungsregisters erfaßt wird, auf Ausgang. Danach ist es möglich, mittels POKE 1,51 auf den DIN- und mittels POKE 1,115 auf den ASCII-Zeichensatz umzuschalten. Sobald man das Datenrichtungsregister wieder auf den Normalwert setzt (durch POKE 0,4,7), schaltet sich wieder der ASCII-Zeichensatz ein. Durch POKE 0,111 bleibt die ASCII/DIN-Taste gesperrt. Das ganze funktioniert nur dann, wenn vor dem POKE 0,111 der ASCII-Zeichensatz eingeschaltet, also die Taste nicht gedrückt war.

Die nächsten acht Häuser dienen vor allem dem Monitor, der ja durch F8 aufrufbar ist. Außerdem benutzt sie der Computer zur Parameterübergabe bei bestimmten Operationen, die sich zwischen verschiedenen BANKs abspielen. Den Basic-Programmierer interessieren sie daher weniger. Hier die Bedeutungen:

Adresse	Name	Aufgabe
02	BANK	BANKteil des Programmzählers PC bei Registeranzeige
03	PC-HI	MSB des Programmzählers bei Registeranzeige
04	PC-LO	LSB des Programmzählers
05	S-REG	Prozessorstatus SR bei Registeranzeige
06	A-REG	Akkumulator A bei Registeranzeige
07	X-REG	X-Register X bei Registeranzeige
08	Y-REG	Y-Register Y bei Registeranzeige
09	STKPTR	Stapelzeiger SP bei Registeranzeige

Bei den nun folgenden Häusern handelt es sich um solche, in denen bestimmte Aufgaben erledigt werden, die mit dem

Basic-Betrieb zusammenhängen. Dabei kommt der Adresse 09 außer der eben vorgestellten noch eine weitere Aufgabe zu: sie enthält ein Zeichen, das im Basic-Text gesucht wird (wie beispielsweise ein Anführungszeichen, einen Doppelpunkt oder ein RETURN zur Beendigung der Basiczeile). All diese Funktionen gab es auch schon beim C64. Dort waren die entsprechenden Speicherstellen genau 2 Byte tiefer gelegen: Der Adresse 09 entspricht beim C64 die Adresse 07 und so fort. Wegen einer genauen Beschreibung verweise ich Sie daher auf den Artikel von Dr. Hauck.

Arrays und Variable

Die Adressen 45 bis 58 gehören zu Zeigern, die ebenfalls im C64 zu finden sind (43 bis 56). Auch hier empfehle ich Ihnen die Lektüre der Memory Map von Dr. Hauck. Die Namen und die Bedeutungen stimmen bei beiden Computern überein (lediglich der letzte Zeiger heißt hier MAXMEM1). Bedingt aber durch den komplexeren Aufbau des C128 finden Sie völlig andere Inhalte.

45/46 ist ein Zeiger, der auf den Start des Basic-Textspeichers weist. Er heißt TXTTAB. Im LSB/MSB-Format enthält er die Adresse $1C01 der BANK 0. Der Basic-Text-Start ist durch Eintragungen in diesen Zeiger verschiebbar, beispielsweise macht das der Computer in Windeseile, wenn ein Bitmap-Modus (also ein Grafikbildschirm) eingeschaltet wird. Dann nämlich findet man hier plötzlich die Basic-Startadresse $4001. Erst durch den Befehl GRAPHIC CLR landet wieder die Anschrift $1C01 in diesem Zeiger. Wollen Sie selbst diese Startadresse verlegen (das muß geschehen, bevor ein Basic-Programm im Speicher liegt!), zum Beispiel nach $2001, dann verfahren Sie so:

1) Berechnen des LSB und des MSB: LSB = $01 = 1, MSB = $20 = 32
2) Eintragen in den Vektor TXTTAB: POKE45,1:POKE46,32
3) NEW eingeben! Damit werden alle anderen Vektoren – soweit nötig – angepaßt.

Durch PRINT FRE(0) können Sie dann schnell feststellen, daß der Speicherplatz in der BANK 0 abgenommen hat. Eine interessante Kleinigkeit noch am Rande: Sollten Sie auf die eben beschriebene Weise einmal den Basic-Textstart verschoben haben und dann durch einen GRAPHIC-Befehl den Bitmap-Modus aktivieren, dann verschiebt der Computer den Basic-Textstart nicht nach $4000, sondern er legt ihn immer um 9216 Speicherplätze höher als die Adresse, die in 45/46 angegeben wurde. Das wäre dann in unserem eben gewählten Beispiel $4401.

47/48 ist wieder ein Vektor. Er heißt VARTAB und weist auf die Adresse, bei der die erste Variable abgelegt wird. Der C128 benutzt dazu im Normalfall die Speicherstelle $0400 in der BANK 1. Auch hier ist es wieder möglich, die Adresse zu verschieben, indem man andere Werte nach 47/48 einträgt. Die Veränderung im Speicher der BANK 1 kann dann mittels PRINT FRE(1) festgestellt werden. Auch hier sollte man nach den POKE-Kommandos ein NEW eingeben. Jede Variable beansprucht in dieser Variablenliste sieben Speicherplätze. Der genaue Aufbau eines Variableneintrages kann bei Dr. Hauck nachgelesen werden, er unterscheidet sich kaum bei beiden Computern.

Interessant in diesem Zusammenhang ist die POINTER-Funktion des Basic 7.0, mit deren Hilfe der Ort in der BANK 1 festgestellt werden kann, an dem ein Variableneintrag steht. Das probieren wir mal aus.

Zuerst löschen wir den Variablenspeicher durch CLR. Dann geben Sie bitte im Direktmodus ein:

```
A% = 32 : B% = 10
```

Ebenfalls im Direktmodus schreiben Sie folgende Zeile:

```
PRINT POINTER(A%),HEX$(POINTER(A%)),POINTER(B%),
HEX$ (POINTER(B%)) <RETURN>
```

Als Ergebnis druckt der Computer aus:

```
1025      0402      1033      0409
```

Die sich ergebenden Werte zeigen mehrerlei: Zum einen erkennt man den Abstand von 7 Byte beider Variableneinträge voneinander. Zum anderen sehen Sie, daß die ausgeworfene Adresse von A% nicht dem vorhin festgestellten Variablenstartwert $0400 entspricht, sondern 2 Byte höher liegt. Der Unterschied rührt vom Variablennamen her, der 2 Byte beansprucht. Das Ergebnis der Pointer-Funktion weist also direkt auf den Wert der Variablen. Mittels des Monitors können Sie das überprüfen: Drücken Sie F8 und geben Sie ein

```
M 10400 10401
```

Eine Zeile wird ausgegeben, von der die beiden ersten Bytes zum Namen gehören (ASCII-Werte + 128: C1 80), die folgenden 5 Byte zum Wert (in der Reihenfolge MSB LSB: 00 20) und zum Füllen (00 00 00). Dann schließt sich die nächste Variable an. Dies ist nicht der geeignete Ort, um tiefer in die Struktur der verschiedenen Variableneinträge einzusteigen (in der Serie »Von Basic zu Assembler« im 64'er-Stammheft geschieht das demnächst). Es sollte nur noch erwähnt werden, daß die POINTER-Funktion bei String-Variablen auf das erste Byte des Stringdeskriptors weist.

Der folgende Zeiger 49/50 bezeichnet den Anfang des Bereiches in der BANK 1, der zur Speicherung von Feldern (auch Arrays oder dimensionierte Variable genannt) dient. Gleichzeitig liegt hier das Ende der einfachen Variableneinträge. Im Einschaltzustand findet man hier den gleichen Eintrag wie in 47/48, nämlich die Adresse $0400. Mit jeder neuen Variablen verschiebt sich dieser Zeiger um 7 Byte aufwärts und mit ihm natürlich auch alle Einträge in der Arrayliste. Stellen Sie sich vor, Sie hätten ein Programm geschrieben, in dem zunächst irgendeine Variable angesprochen wird, dann mehrere Felder durch DIM... definiert werden. Im weiteren Programmlauf taucht noch die eine oder andere Variable auf: Schleifenzähler, Zwischenwerte und so weiter. Jede neu auftretende Variable führt dazu, daß der gesamte Arrayinhalt um 7 Byte höher gelegt werden muß. Sie wissen vermutlich auch schon, daß Arrays viel Speicherplatz fressen und können sich daher nun auch vorstellen, daß diese Verschiebungen Zeit kosten. Die Lehre aus dieser Erkenntnis führt zu einer Beschleunigung des Programms: Jede Variable sollte schon definiert werden (unter Umständen mit irgendwelchen Dummywerten), bevor das erste Array definiert wird. Auf diese Weise braucht nichts mehr verschoben zu werden.

Variable vordefinieren!

Der Name des Arraystartzeigers ist übrigens ARYTAB und auch hier lassen sich die Orte der einzelnen Einträge mittels der POINTER-Funktion feststellen. Das sollten Sie mal mit einem zweidimensionalen Array ausprobieren, um festzustellen, in welcher Reihenfolge die einzelnen Elemente abgelegt sind.

Der Zeiger 51/52 heißt STREND und gibt die Endadresse der Arrayeinträge an. Er wächst also mit jedem neudefinierten Feld um die Anzahl Byte weiter, die das Feld für alle Elemente und für den Kopf benötigt. Sehen wir uns nun einmal an, wieviele Byte das sind. Geben Sie zuerst CLR ein, um den Variablenspeicher zu löschen. Im Direktmodus schreiben Sie dann:

```
DIM AB(300,32)
```

Mittels des Monitorkommandos M 10400 10401 lassen Sie sich wieder eine Zeile aus dem Variablenspeicher auf

dem Bildschirm ausgeben. Weil wir nur das Array definiert haben, gehört alles, was wir nun sehen, zum Array:
41 42 0A C2 02 00 21 01 2D 00 ...

Dies ist der Kopf unseres Feldes. Die beiden ersten Angaben kennzeichnen den Namen und den Typ des Arrays (41 42 = ASC II für A B). Die beiden folgenden Bytes geben im LSB/MSB-Format die Länge unseres gesamten Feldes an: $C20A = 49674! Ganz schön lang, unser Feld. Es folgt die Anzahl der Dimensionen (02): in unserem Beispiel ist diese Anzahl 2. Weil ein ganzes Byte für diese Zahl reserviert wird, könnte man theoretisch bis zu 255 Dimensionen verwenden. Bis hierher ist der Aufbau des Arraykopfes für alle möglichen Felder identisch. Nun folgen – jeweils in 2 Byte angegeben – die Anzahl der Elemente pro Dimension. Dabei fängt die Eintragung bei der letzten angegebenen Dimension an: $0021 gehört zu der von uns eingegebenen Menge von 32 Elementen in der zweiten Dimension. $21 entspricht der dezimalen Angabe 33. Das kommt dadurch zustande, daß auch ein Element mit der Nummer 0 mitgezählt wird. Vielleicht ist es Ihnen schon aufgefallen, daß diese Angabe der Elementanzahl im ungewöhnlichen Format MSB/LSB stattfindet. Das sehen Sie auch an der nächsten Elementangabe $012D, was dezimal 301 entspricht. Wir fassen also zusammen:

Fester Arraykopf : 5 Byte
Für jede Dimension : 2 Byte

In unserem Beispiel ergibt sich daher eine Länge von 5+2*2 = 9 Byte. An diesen Kopf schließen sich nun die einzelnen Elementeinträge an. Dabei haben je nach Arraytyp die Einträge unterschiedliche Längen:

Integer-Array : 2 Byte
Fließkomma-Array : 5 Byte
String-Array : 3 Byte

Unser Beispielfeld ist ein Fließkomma-Array. Jeder Eintrag beansprucht daher 5 Byte. Die Anzahl der Elemente (mit den jeweiligen Null-Einträgen) ist (32+1)*(300+1) = 9933. Diese Anzahl also mal fünf pro Eintrag ergibt 49665 Byte für alle Einträge zusammen. Dazu addieren wir noch die 9 Byte für den Kopf und erhalten so die Gesamtlänge von 49674 Byte. Das hatten wir oben auch schon aus den Byte 3 und 4 des Arraykopfes herausgelesen.

Der Vektor 51/52 sollte nun den Wert $0400 + $C20A = $C60A enthalten. Sehen Sie nach mittels des Monitorkommandos M 00033 00034: genau an den ersten beiden Stellen steht im LSB/MSB-Format diese Adresse. Sie wissen nun, wie man die Länge eines Feldes berechnen kann. Gleichzeitig haben Sie aber auch gesehen, daß solch ein Feld sehr schnell eine gewaltige Größe annehmen kann. Das sollten Sie im Gedächtnis behalten, denn bei den nächsten Vektoren stoßen wir nochmal auf diese Eigenschaft.

FRETOP wird der nächste Vektor bei 53/54 genannt. Er weist immer auf das Ende des Bereiches, in dem Stringtexte stehen. Im Einschaltzustand findet man hier die Adresse $FF00 in BANK 1. Definiert man nun im Programm oder im Direktmodus einen Stringtext, dann lagert der Interpreter unseres Computers diesen Text und noch einen Zeiger (der besteht aus 2 Byte und weist auf den Stringdeskriptor) von $FF00 an abwärts in den Speicher. Der Vektor 53/54 folgt dieser Eintragung und sein Inhalt nimmt einen geringeren Wert an. Bestünde der einzutragende String aus 4 Zeichen, dann senkt sich 53/54 um 6 Byte. Wenn viele und lange Strings auftreten, dann nähert sich – immer weiter abwärts schreitend – der Vektor 53/54 langsam aber sicher dem Vektor 51/52. Vor jedem neuen Eintrag prüft unser Computer, ob noch genug Platz zwischen den beiden Adressen vorhanden ist. Reicht einmal dieser Raum nicht mehr aus, dann erfolgt die Garbage Collection. Darunter versteht man die Vernichtung von nicht mehr benötigten Stringtexten, die dazu führt, daß der Abstand beider Vektoren wieder etwas wächst. Sollte aber irgendwann einmal auch diese Operation nicht mehr genügend Raum schaffen können, dann meldet sich der Computer mit einem OUT OF MEMORY ERROR. Beim C 128 dürfte das allerdings nicht allzu häufig der Fall sein.

Der Vektor 55/56 heißt FRESPC und ist ein für uns nicht so bedeutender Hilfszeiger zur Stringverarbeitung.

Interessant ist wieder MAXMEM1, wie der Zeiger 57/58 genannt wird. Er enthält die höchste für die Variablenspeicherung zur Verfügung stehende Adresse in der BANK 1. Der normale Inhalt ist $FF00. Es gibt Situationen, in denen es wünschenswert erscheint, den oberen Speicherraum der BANK 1 für andere Zwecke zu benutzen. In solchen Fällen genügt es dann, die neue Variablenspeichergrenze im LSB/MSB-Format in MAXMEM1 einzuschreiben und anschließend CLR einzugeben. Durch das letztere Kommando werden auch die anderen Zeiger auf sinnvolle Werte gesetzt.

Wo hört Basic auf?

Wir müssen nun der Reihenfolge der Häuser in der Zeropage-Straße gewaltig vorausgreifen: Nachdem wir Vektoren gefunden haben, die jede Einzelheit der Variablenspeicherung in der BANK 1 fixieren, die das in Basic erreichbare Ende dieser Bank festlegen und die den Beginn des Basic-Textes in BANK 0 kennzeichnen, fehlen uns noch zwei Angaben: Wo hört der Basic-Text auf und wo ist das für den Basic-Text maximal erreichbare Ende des Speichers in der BANK 0?

Dazu eilen wir in die Page 12 (kann man das noch als erweiterte Zeropage bezeichnen?) und finden darin den Vektor 4624/4625 (das ist $1210/1211), der TEXTTOP genannt wird. Darin befindet sich die Endadresse des Basic-Programmtextes. Ebenfalls in dieser Page liegt der Zeiger MAXMEM0, nämlich bei 4626/4627 ($1212/1213). Wie es der Name schon nahelegt, bezeichnet dieser Vektor das äußerste Ende des Basic-Textspeichers in BANK 0. Auch hier gibt es wieder die Möglichkeit, durch geeignete POKE-Befehle diese Grenze herabzusetzen. Mit dieser Kenntnis der entscheidenden Basic-Vektoren wird es möglich, Assembler-Programme zu schreiben, die einen OLD- und einen MERGE-Befehl enthalten. Beide Programme finden Sie im Buch »Grafik-Programmierung C 128«, (auf den Seiten 144 und 150) das im Markt & Technik Verlag unter der Nummer MT90202 erschienen ist oder in den 64'er-Ausgaben 12/85, Seite 43 und 5/86, Seite 95.

Das Bild 2 zeigt Ihnen die Bedeutung all dieser Zeiger für die BANK 0 und BANK 1 in übersichtlicher Form.

Direkt nach der Gruppe von Basic-Zeigern finden wir den Vektor CURLIN (59/60), der dasselbe leistet wie der Vektor 57/58 im C 64. Daher verweise ich Sie wieder an die Memory Map von Dr. Hauck.

TXTPTR bei 61/62 ist aber etwas Neues. Das ist ein Vektor, der in der sogenannten CHRGET-Routine verwendet wird und immer auf das nächste Zeichen im Basic-Text weist. Für den Basic-Programmierer allerdings dürfte ein Verändern dieser beiden Speicherstellen auf das bekannte Absägen des Astes, auf dem man sitzt, hinauslaufen. Die CHRGET-Routine nämlich arbeitet sich Zeichen für Zeichen durch den Basic-Text und schaufelt auf diese Weise dem Basic-Interpreter den Text Byte für Byte zu. Wird also im Rahmen eines Programms TXTPTR durch das Programm selbst verändert, dann kann schon nach dem ersten POKE-Befehl (beispielsweise nach 61) der Computer den zweiten POKE-Befehl nicht mehr finden (den nach 62), weil der Zeiger nun schon ganz woanders hinweist.

Bei 63/64 handelt es sich um den Hilfszeiger FNDPNT, der unter anderem beim PRINT USING eine Rolle spielt.

Die Bedeutung der Speicherstellen 65 bis 98 ist identisch mit der der um 2 erniedrigten Adressen im C 64 (also 63 bis

96). Weitere Informationen dazu finden Sie bei Dr. Hauck.

Auch die folgenden Speicherstellen 99 bis 104 und 106 bis 111 enthalten dasselbe, wie die C 64-Speicherstellen 97 bis 102 und 105 bis 110: hier liegen die beiden Fließkomma-Akkumulatoren FAC und ARG. Bild 3 zeigt den Aufbau dieser beiden Speicherbereiche.

Die Fließkomma-Akkumulatoren sind gewissermaßen die Hauptrechenwerke unseres Computers, der nahezu alle Rechenoperationen mit Fließkommazahlen ausführt. Hier müssen vor einem Aufruf einer mathematischen Operation die davon betroffenen Zahlen enthalten sein. Nach der Operation findet sich das Ergebnis fast immer im FAC. Für den Basic-Programmierer haben die beiden Akkumulatoren kaum eine direkte Bedeutung, dafür sind sie für den Assemblerprogrammierer – besonders für den, der auch Interpreterroutinen verwendet – das tägliche Brot.

FAC und ARG

Eine Speicherstelle haben wir bislang ausgelassen: SGNFLG bei 105. Dieses Byte liegt beim C 64 bei 103 und dort erfüllt es auch die gleiche Aufgabe bei Polynomauswertungen. Die Speicherstellen 112 (ARISGN) und 113 (FACOV) entsprechen den C 64-Adressen 111 und 112. Erstere dient dem Vergleich der Vorzeichen von FAC und ARG, letztere ist ein Rundungsbyte für den FAC. In 114/115 liegt ein Zeiger auf den Kassettenpuffer, der im Normalfall ab $B00 zu finden ist. Der Name dieses Vektors ist FBUFPT und im C 64 befindet sich dasselbe bei 113/114. Siehe also dazu wieder den guten Dr. Hauck in seiner Memory-Map-Serie.

Nun kommen wir zu allerlei Unterschieden zwischen dem C 64 und dem C 128. Beim ersteren begann ab 115 die vorhin schon erwähnte CHRGET-Routine. Der C 128 enthält hier statt dessen eine Menge Vektoren und gespeicherter Werte:

116/117 AUTINC enthält die Schrittweite, in der beim AUTO-Befehl die Basic-Zeilennummern erhöht werden.

118 MVDFLG ist ein Flag, das anzeigt, ob durch einen GRAPHIC-Befehl die Verschiebung des Basic-Programmtextes von $1C00 nach $4000 veranlaßt worden ist. In dem Fall ist dieses Flag gesetzt.

119 SPRNUM dient als Mehrzweckspeicher für vielerlei Anwendungen. Beispielsweise enthält er beim MOVSPR-Befehl die Spritenummer (daher der Name), aber auch die Lautstärke beim VOL-Befehl oder andere Dinge werden hier gelagert.

120 HULP wird nur in Verbindung mit SPRNUM gebraucht und hat dann auch vielerlei Funktionen.

121 SYNTMP ist ein Zwischenspeicher, der meist bei Vergleichen mit Werten aus anderen BANKs verwendet wird. Diese sind dann hier abgelegt.

122 DSDLEN dient im Monitorbetrieb als Zeiger in den Eingabepuffer ab $200. Im Basic-Betrieb findet sich hier die Länge von DS$. Die Floppy-Statusvariable wird dann als gelöscht angesehen, wenn hier eine Null enthalten ist.

123/124 DSDADR zeigt in den Stringbereich der BANK 1. Dort befindet sich dann der laufend aktualisierte Wert von DS$.

125/126 TOS ist ein Zeiger in den Basic-Programmlauf-Stapelspeicher (der geht von $A00 bis $800). Hier werden während eines Programmes Informationen über FOR...NEXT, DO...LOOP-Schleifen oder Return-Adressen aufbewahrt. Beim C 64 geschah das noch im Prozessor-Stapelspeicher, der aber für die stark erweiterten Möglichkeiten des Basic 7.0 zu klein wäre.

127 RUNMOD enthält Informationen darüber, ob sich der Computer im Direkt- (dann liegt hier $00) oder im Programmmodus (Inhalt $80) befindet. Das entspricht der MSGFLG bei 157 des C 64.

128 SYNTAX1 enthält ein Prüfbyte der sogenannten DOSPAR-Routine. Diese Routine holt die Parameter für Diskettenbefehle. Die Bedeutung der einzelnen Bits ist:

Bit	Zuordnung
0	1. Filenamen gelesen
1	2. Filenamen gelesen
2	logische Filenummer gelesen
3	Gerätenummer gelesen
4	1. Laufwerksnummer gelesen
5	2. Laufwerksnummer gelesen
6	Recordlänge gelesen
7	Klammeraffe gelesen (Überschreiben des Files)

Ebenfalls von dieser Routine angesteuert wird die folgende Speicherstelle:

129 SYNTAX2. Hier haben nur die Bits 0 bis 2 eine Bedeutung:

Bit	Zuordnung
0	Segmentnummer gelesen
1	Startadresse gelesen
2	Endadresse gelesen

130 OLDSTK dient beim TRAP-Befehl des Basic 7.0 als Aufbewahrungsort für den Stapelzeiger des Prozessors.

Bevor wir hier weiter untersuchen, noch eine kleine Bemerkung: Sie haben sicherlich schnell festgestellt, daß wir das Bild von der Speicherstadt mit den Häusern der Zeropage-Straße oft fallengelassen haben. Das hängt damit zusammen, daß die Materie stellenweise recht kompliziert ist und man davon ausgehen kann, daß sowohl Einsteiger als auch Freaks diese C 128-Memory-Map verwenden. Es wäre unsinnig, beispielsweise die gerade vorgestellte Speicherstelle 130 als Haus zu bezeichnen, wenn man nun nicht auch noch Entsprechungen für den Begriff des Stapelzeigers oder des Prozessors verwenden würde. Das aber würde sehr auf Kosten der Klarheit gehen und ab und zu auch reichlich lächerlich klingen. Ohnehin kann mit einigen Speicherstellen nur der Fortgeschrittene etwas anfangen. Deshalb wird unser Bild von der Speicherstadt auch nur dort verwendet, wo es darum geht, dem Einsteiger einen Zusammenhang zu verdeutlichen.

131 COLSEL enthält im Einschaltzustand den Wert 0. Hier ist gespeichert, welche Farbquelle bei einer grafischen Operation (beispielsweise DRAW) herangezogen wird. Folgende Möglichkeiten stehen zur Verfügung:

0 Hintergrundfarbe
1 Vordergrundfarbe
2 Multicolorfarbe 1
3 Multicolorfarbe 2

Auf diese Weise ist es möglich, daß Grafikbefehle auch ohne Angabe einer Farbquelle gegeben werden. In diesem Fall wird die in COLSEL gespeicherte Farbquelle angesprochen, also die aus der letzten grafischen Operation.

Multicolor-Farben

132 MCOLOR1 enthält die aktuelle Multicolorfarbe 1. Im Einschaltzustand findet sich hier der Wert 1, was dem Farbcode von Weiß (1) entspricht.

133 MCOLOR2 weist bei Einschalten den Wert 2 auf (Rot – 1) und dient der Aufbewahrung der aktuellen Multicolorfarbe 2.

134 FOREGND ist im Einschaltzustand auf den Wert 13 (Hellgrün – 1) gesetzt und enthält den Code der Vordergrundfarbe.

135/136 SCALEX dient zur Aufbewahrung des maximalen X-Wertes beim SCALE-Befehl. Interessant scheint, daß man – entgegen der Angaben im Handbuch – hier Werte bis 32767 (der höchsten positiven 16-Bit-Integerzahl) verwenden kann.

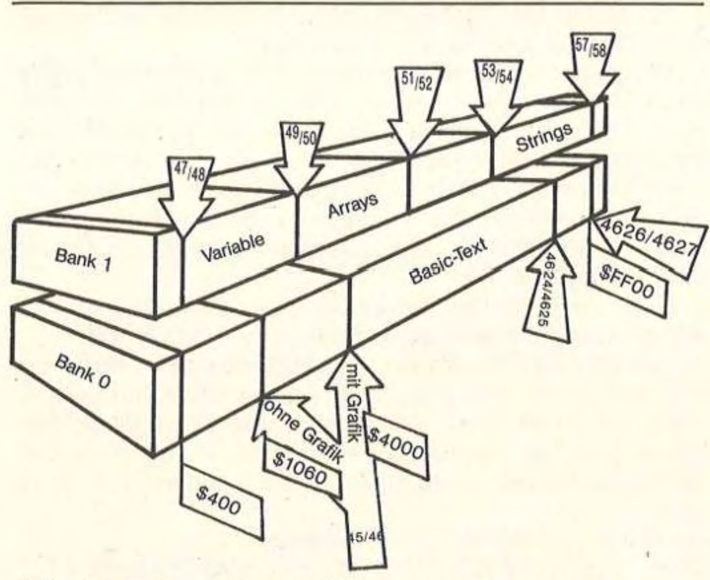


Bild 2. Die Zeiger in das Basic-Text- und -Variablen-RAM in BANK 0 und BANK 1

FAC

Byte	99	100	101	102	103	104
Inhalt	Exponent inklusive Vorzeichen	MANTISSE				Vorzeichen der Mantisse

ARG

Byte	106	107	108	109	110	111
Inhalt	Exponent inklusive Vorzeichen	MANTISSE				Vorzeichen der Mantisse

Bild 3. Anordnung und Inhalte der Fließkomma-Akkumulatoren FAC und ARG

Bit:	3	2	1	0			
	R	G	B	J	Dez.	Farbe	Entspricht Farbcode im 40-Z.-Modus
	0	0	0	0	0	schwarz	1
	0	0	0	1	1	dunkelgrau	12
	0	0	1	0	2	blau	7
	0	0	1	1	3	hellblau	15
	0	1	0	0	4	grün	6
	0	1	0	1	5	hellgrün	14
	0	1	1	0	6	türkis	4
	0	1	1	1	7	orange	9
	1	0	0	0	8	rot	3
	1	0	0	1	9	hellrot	11
	1	0	1	0	10	lila	13
	1	0	1	1	11	violett	5
	1	1	0	0	12	braun	10
	1	1	0	1	13	gelb	8
	1	1	1	0	14	hellgrau	16
	1	1	1	1	15	weiß	2

Bild 4. Aufbau-Schema der Speicherstelle 241 (80-Zeichen-Betrieb)

Bit:	7	6	5	4	3	2	1	0
Name:	ALT	RVS	UL	FL	R	G	B	J

Bild 5. Kombinationen der Bits 0 bis 3 der Speicherstelle 241 und die sich ergebenden Farben (Sehr von der Art und Einstellung des verwendeten 80-Zeichen-Bildschirmes abhängig)

137/138 SCALEY erfüllt den gleichen Zweck wie der eben genannte Speicher, hier aber für den maximalen Y-Wert. Auch bei Y ist es möglich, Werte bis 32767 sinnvoll anzuwenden.

139 STOPNB enthält ein Flag, das beim PAINT-Befehl der Randbegrenzung dient.

140/141 GRAPNT wird verschieden genutzt als Zeiger in die Bitmap.

142 VTEMP1 und 143 VTEMP2 sind Zwischenspeicher für verschiedene Grafikoperationen.

Zeropageadressen des Betriebssystems

Die Speicherstellen 144 STATUS bis 203 ROBUF kann man als Zeropage des Betriebssystems bezeichnen. Größtenteils sind hier die Zuordnungen – bei gleichen Adressen – identisch mit denen im C 64. Lesen Sie für weitere Informationen dazu in dem Kurs über die Memory Map nach. Hier sollen nun nur die Unterschiede – soweit bekannt – erklärt werden.

145 STKEY hat zwar dieselbe Aufgabe wie bei Dr. Hauck beschrieben, basiert aber auf einer veränderten Tastaturmatrix. Es gibt ja wesentlich mehr Tasten beim C 128. Die Matrix besteht in unserem Computer aus 11 Zeilen zu je 8 Plätzen und ist – wie beim C 64 – mit den Ports A (Zeilen) und B (Spalten) des CIA 1 verbunden. Weil aber durch diese Ports nur je 8 Plätze erfaßt werden können, hat ein neues Register 47 des VIC-Chip die Aufgabe übernommen, mit seinen Bits 0 bis 2 die Zeilen 9 bis 11 der Tastaturmatrix zu beobachten. Weil aber in diesen drei zusätzlichen Zeilen ausschließlich die neuen Tasten abgefragt werden (der Zehnerblock, ESC, TAB, ALT, HELP, LINE FEED, NO SCROLL und obere Cursortasten), und weil die Anordnung der anderen Tasten in der Matrix identisch mit der im C 64 ist, ergeben sich in 145 die gleichen Werte wie bei Dr. Hauck beschrieben.

147 VERCK verhält sich genauso wie beim C 64. Sein Inhalt ist identisch mit dem der Speicherstelle 12 (nicht 10 wie beim C 64).

172/173 SAL hat zwar die bei Dr. Hauck beschriebene Funktion, außerdem aber liegt hier die Startadresse eines Programmes beim BOOT-Befehl, wenn mit der Floppy 1570 oder 1571 gearbeitet wird.

191 DRIVE hat außer der Aufgabe der Speicherstelle MYCH des C 64 – die an derselben Stelle liegt – noch zusätzlich eine beim Booten: Der ASCII-Wert »0« liegt hier für das Laufwerk parat.

193/194 STAL enthält außer den bei Dr. Hauck beschriebenen Werten beim Booten noch Track (in 194) und Sektor (in 193) des Programmes.

195/196 MEMUSS dient auch im C 128 als Zeiger auf einen Programmanfang bei LOAD und VERIFY. Außerdem ist MEMUSS aber noch ein Zwischenspeicher beim Kopieren der Kernal-Vektoren in die erweiterte Zeropage und bei der Suche nach der Zeichenfolge »CBM« in der BANK 1.

197 DATA ist ein Bitspeicher für das Lesen vom Band und auch ein Prüfsummenpuffer für das Schreiben auf Band.

198 BA steuert bei LOAD, SAVE und VERIFY die Speicherkonfiguration, auf die zugegriffen wird.

199 FNBANK erledigt dasselbe für den Zugriff auf einen aktuellen Filenamen.

200/201 RIBUF gibt die Startadresse des RS232-Eingabepuffers an, der von $C00 bis $D00 in BANK 0 reicht. Er entspricht etwa dem Zeiger 247/248 des C 64.

202/203 ROBUF enthält die Anfangsadresse des RS232-Ausgabepuffers. Dieser liegt zwischen $D00 und $E00 in der BANK 0. Ein ähnlicher Zeiger findet sich im C 64 bei 249/250.

204/205 KEYTAB ist ein Zeiger auf die aktuelle Tastaturdecodierertabelle. Dabei wird ein Wert aus der Zeigertabelle ab 830 verwendet.

206/207 IMPARM ist ein Vektor, der für die Kernal-PRIMM-Routine reserviert ist. Für Assembler-Programmierer ist diese Routine, die durch JSR $FF7D aufgerufen werden kann und einen direkt anschließenden Text ausgibt, sehr interessant.

208 NDX entspricht der C 64-Speicherstelle 198. Hier ist die Anzahl der Zeichen im Tastaturpuffer (liegt von 842 bis 851) enthalten. Vorgesehen ist – wie auch schon beim C 64 – die Speicherung von 10 Zeichen. Allerdings nimmt es der C 128 nicht übel, wenn man noch weitere 10 benutzt, lediglich die Tabulatorstopps, die durch CTRL-I oder TAB aufgerufen werden, hat er dann vergessen. Die Verwendungsmöglichkeiten dieser Speicherstelle und des Tastaturpuffers im sogenannten programmierten Direktmodus werden Sie später erkennen, wenn wir uns den Tastaturpuffer genauer ansehen.

209 KYNDX enthält die Länge eines Funktionstasten-strings. Gleichzeitig dient diese Speicherstelle als Flag für das Holen von Zeichen aus dem String, wenn sie ungleich Null sind.

210 KEYIDX ist ein 1-Byte-Zeiger auf den aktuellen Funktionstastenstring, der in den Speicherbereich ab $100A weist.

211 wird SHFLG genannt und enthält einen Code für bestimmte Tasten:

Bit	Taste
0	Shift
1	CBM-Taste
2	Control-Taste
3	ALT 4 DIN

Jedesmal, wenn solch ein Bit gleich 1 ist, ist die entsprechende Taste gedrückt. Ist der Inhalt gleich 0, dann ist keine von diesen Tasten betätigt worden.

212 enthält die Codenummer der gerade gedrückten Taste und wird SFDX genannt. Welche Taste zu welcher Nummer gehört, können Sie aus dem Handbuch, Anhang J, entnehmen.

213 LSTX speichert ebenfalls den Tastencode. Diese Speicherung dient der Repeat-Funktion.

Ein Flag zum Lesen vom Bildschirm liegt in 214 (CRSW). Ist dieses Flag ungleich Null, dann wird bei eingeschaltetem Cursor gelesen.

215 MODE gibt an, welcher Bildschirm (40 oder 80 Zeichen) gerade der Ausgabebildschirm ist. Man findet hier den Inhalt 0, wenn der 40-Zeichen- und den Inhalt 128, wenn der 80-Zeichen-Bildschirm aktiv ist. Das ist für die Textausgabe auf dem Bildschirm wichtig. Es genügt eine Basic-Zeile wie:

```
IF PEEK(215) > 0 THEN ... : ELSE ...
```

216 GRAPHM gibt Auskunft über den aktuellen Grafikmodus. Ein bequemerer Weg, diesen zu erfahren, ist der Basic-Befehl RGR. Man findet in 216 folgenden Zusammenhang:

Inhalt	Modus
0	Text-Modus
32	HiRes-Bitmap-Modus
96	Splitscreen: HiRes und Text
160	Multicolor-Modus
224	Splitscreen: Multicolor und Text

217 CHAREN hat eine noch unklare Bedeutung: In der Literatur finden sich zwei Beziehungen. Zum einen soll diese Speicherstelle als Flag für den Rastervergleichs-Interrupt dienen. Zum anderen aber soll der Inhalt des Bit 1 angeben, ob Zeichen aus dem ROM (=0) oder RAM (=1) entnommen werden. Das gilt immer dann, wenn über den VIC-Chip Zeichen ausgegeben werden. Allerdings kann man auch ohne Berücksichtigung dieser Speicherstelle eigene Zeichen aus dem RAM holen.

218/219 SEDSAL und 220/221 SEDEAL sind zwei Hilfszeiger des Bildschirmeditors, die beispielsweise für das Scrollen oder Löschen von Zeilen nötig sind.

222 SEDT1 und 223 SEDT2 sind ebenfalls Hilfszellen, die als 1-Byte-Speicher des Editors dienen.

Die nachfolgenden Speicherstellen sind praktisch zweimal vorhanden, nämlich einmal für den 40-Zeichen- und dann noch für den 80-Zeichen-Bildschirm. Der jeweils aktuelle Inhalt findet sich in den nun folgenden Speicherstellen, wohingegen die Parameter des anderen Bildschirmes ab 2624 gelagert sind. Wechselt man den Bildschirm (beispielsweise durch ESC-X), dann werden blitzartig die beiden Inhalte ausgetauscht.

224/225 PNT ist ein Zeiger, der die Adresse der aktuellen Bildschirmzeile im jeweiligen Bildschirm-RAM enthält.

226/227 USER enthält ebenfalls solch einen Zeiger, nun aber in das Farb-RAM beziehungsweise in das Attribut-RAM.

Die folgenden vier Speicherstellen haben mit Bildschirmfenstern zu tun. Als Fenster kann auch der gesamte Bildschirm verstanden werden. Es gelten folgende Zuordnungen:

Adresse	Name	Funktion
228	SCBOT	Unterer Rand (Maximal 24)
229	SCTOP	Oberer Rand (Minimal 0)
230	SCLF	Linker Rand (Minimal 0)
231	SCRT	Rechter Rand (Maximal 39 oder 79)

Die in Klammern angegebenen Werte beziehen sich jeweils auf den ganzen Bildschirm als Fenster. Der WINDOW-Befehl schreibt in diese Speicherstellen, der RWINDOW-Befehl liest sie aus.

232 LSXP und 233 LSTP enthalten die Startspalte und die Startzeile für Eingaben bei blinkendem Cursor.

234 INDX gibt das Ende der Zeile bei einer Eingabe an.

235 TBLX und 236 PNTR halten die aktuelle Cursorzeile und Cursorspalte fest.

In 237 LINES befindet sich die höchste zulässige Zeilennummer, also 24.

238 COLUMNS enthält die höchste zulässige Spaltennummer, die je nach aktuellem Bildschirm 39 oder 79 beträgt. Auch hier ist eine Prüfung auf den gerade eingesetzten Bildschirm möglich.

239 DATAX enthält das nächste auszugebende Zeichen.

240 LSTCHR dagegen speichert das zuvor ausgegebene Zeichen. Diese Speicherung ist nötig, um beispielsweise ESC-Sequenzen zu bearbeiten.

241 COLOR enthält – je nach angeschlossenem Bildschirm – Zeichenfarbe oder Attribut des auszugebenden Zeichens. Beim 40-Zeichen-Bildschirm entspricht diese Speicherstelle der bei 646 im C 64. Ihr Inhalt ist dann gleich dem Farbcode 1 (also beispielsweise 1 für Weiß). Die Möglichkeiten bei angeschlossenem 80-Zeichen-Bildschirm sind weitaus vielfältiger. Bild 4 zeigt Ihnen den Aufbau von COLOR, der gleich dem einer Attribut-RAM-Zelle des VDC-Chip unseres Computers ist.

Bit 7: Im Gegensatz zum 40-Zeichen-Bildschirm, der die Umschaltung zwischen den beiden Zeichensätzen (Groß- und Kleinschreibung) nur durch Umschalten (per Commodore- und Shift-Taste) erlaubt, können auf dem 80-Zeichen-Bildschirm beide Zeichensätze gleichzeitig verwendet werden. Welches von den beiden möglichen Zeichenmustern abgebildet wird, entscheidet dieses Bit. Ist hier eine 1 enthalten, dann stammt das Zeichen aus dem zweiten Zeichensatz.

Bit 6: Hiermit wird ein Aspekt der Zeichendarstellung doppelt gemoppelt: die Reverse-Darstellung. Eigentlich kann man nämlich durch Eintragen einer 1 in dieses Bit genau dasselbe erreichen, wie durch das ansonsten etwas speicherfressende Verfahren eines gesonderten Zeichenmustersatzes, aus dem sonst die inversen Zeichen abgerufen werden.

Bit 5: Das Zeichen ist unterstrichen, wenn sich hier eine 1 befindet.

Bit 4: Wenn dieses Bit gleich 1 ist, dann blinkt das Zeichen.

Bits 3 bis 0: Diese Bits regeln durch ihre Kombination die dargestellte Farbe. R steht für Rot, G für Grün, B für Blau und I für Intensität.

Alle diese einzelnen Aspekte können durch CHR$- oder/und ESC-Befehle gesteuert werden. So erzeugt:

```
PRINT CHR$(2)"A"
```

ein unterstrichenes und

```
PRINT CHR$(27)+"F";"A"
```

ein blinkendes A. Das Handbuch gibt ab Seite 44 darüber Auskunft. Aber auch durch einen sinnvollen POKE-Befehl in diese Speicherstelle 241 können Sie alle Optionen ansteuern. Nehmen wir an, daß wir unterstrichene blinkende blaue Zeichen ausgeben möchten, also die Bits 5,4 und 1 auf 1 setzen müssen:

```
N = 0*2↑7 + 0*2↑6 + 1*2↑5 + 1*2↑4 + 0*2↑3
    + 0*2↑2 + 1*2↑1 + 0*2↑0
POKE 241,N
```

242 TCOLOR ist ein Zwischenspeicher.

243 RVS enthält ein Reverse-Flag. Steht hier eine 0, dann wird das nächste Zeichen normal ausgegeben, bei anderen Inhalten dagegen invers.

244 QTSW dient als Flag für den sogenannten Quote-Modus. Das ist der Modus, in dem Steuerzeichen als reverse Zeichen auf dem Bildschirm auftreten. Dieser Modus ist eingeschaltet, wenn sich in 244 ein Inhalt ungleich 0 befindet. Durch einen nachfolgenden Print-Befehl können dann Steuerzeichen ausgedruckt statt ausgeführt werden.

245 INSRT zählt im Insert-Betrieb die Anzahl der eingegebenen Inserts.

246 INSFLG ist ein Flag für den automatischen Einfüge-Modus (das ist der Insert-Modus). Dieser Modus wird durch ESC-A eingeschaltet und erzeugt dann in INSFLG einen Wert von 255.

247 wird LOCKS genannt und sperrt einige Tastenfunktionen. Nur die Bits 6 und 7 spielen eine Rolle:

Bit	sperrt bei Inhalt 1
7	Umschaltung durch CBM- und Shift-Taste
6	No-Scroll-Taste und Control-S

248 SCROLL bewirkt, daß ein Aufwärtsscrollen des Bildschirmes verhindert wird, wenn das Bit 7 gesetzt ist. Erreicht dann der Cursor den unteren Bildschirmrand, wird der Bildschirm nicht nach oben gerollt, sondern der Cursor an den oberen Bildschirmrand gesetzt. Das Scrollen durch andere Ursachen (beispielsweise ESC-V) wird dadurch nicht verhindert. Ein gesetztes Bit 6 verhindert, daß die letzte Bildschirmzeile versehentlich ein Zeilenüberlaufbit erhält.

249 BEEPER ist eine Flagge, die das Sperren des Tones durch Control-G ermöglicht. Dazu muß das Bit 7 gleich 1 sein.

250 bis 255 FREKZP: Hier hat man als Assembler-Programmierer einige freie Zeropage-Speicherstellen zur Verfügung. Die Speicherstelle 255 allerdings hat noch den Namen LOFBUF und scheint für irgendwelche Zwecke doch noch vom Computer benutzt zu werden (wie es auch beim C 64 der Fall ist). In der Literatur findet sich aber keine genauere Angabe darüber.

Was nun folgt, könnte man als den erweiterten Teil der Zeropage-Straße bezeichnen. Haben Sie eine Vorstellung davon, wie lang diese »erweiterte Zeropage« ist? Sie reicht bis zur Adresse 4863. Etwa 18 weitere Pages liegen also noch vor uns! Diese lange Straße wird unser Thema sein in einer demnächst beginnenden Artikelfolge im 64'er-Magazin.

(Heimo Ponnath/dm)

LITERATUR:

Dr. H. Hauck: Memory Map mit Wandervorschlägen, Alle Folgen aus dem 64'er-Magazin.

H. Ponnath: Grafik-Programmierung C128, Markt & Technik Verlag München 1986, MT 90202.

R. Schineis, U.M.Braun, N.Demgensky: C 128 ROM-Listing: Operating System, Markt & Technik Verlag München 1986, MT90221.

R.Schineis, M.Braun: C 128 ROM-Listing: Basic-7.0-Betriebssystem, Markt & Technik Verlag München 19[illegible] MT90220.

G.Möllmann: C 128-Programmieren in Maschinensprache, Markt & Technik Verlag München 1986, MT90213.

Spielen mit Musik und Grafik

Lernen Sie das Programmieren von Grafik und Sound auf Ihrem C 64. Daß das auch in Basic möglich ist, beweist der folgende Artikel.

Wie oft schon hat der Basic-Programmierer die hervorragenden Sound- und Grafikmöglichkeiten seines C 64 bestaunt, wenn er ein gekauftes Spiel, Grafik- oder Musikprogramm auf seinem Heimcomputer laufen ließ. Und wie oft hat er sich dabei gewünscht, ebensolche Klänge und Grafiken in seinen eigenen Basic-Programmen verwenden zu können.

Um sich diesen Wunsch zu erfüllen, gibt es zwei Möglichkeiten. Zum einen gibt es da die meist recht teueren Basic-Erweiterungen, die mit Zusatzbefehlen ohne großen Aufwand Sound- und Grafikprogrammierung in Basic möglich machen.

Nachteilig dabei ist jedoch, daß ein mit diesen speziellen Befehlen geschriebenes Programm nur mit der verwendeten Basic-Erweiterung lauffähig ist, da für den Standard-Basic-Interpreter zum Beispiel Befehle wie »PLOT« oder »MUSIC« Fremdwörter sind. Sie sind also zum normalen C 64-Basic vollkommen inkompatibel.

Die andere Möglichkeit ist die Programmierung mit dem Basic, das Ihrem C 64 zur Verfügung steht. Sie ist zwar ungleich komplizierter und, wenn es auf Geschwindigkeit ankommt, sehr viel langsamer. Die so geschriebenen Programme sind aber ohne Erweiterungen auf jedem C 64 zu betreiben.

Um die Programmierung von Sound und Grafik in Basic soll es auch in diesem Bericht gehen.

Zu diesem Zweck werden die dafür verantwortlichen C 64-Bausteine, der VIC 6569 (Grafik) und der SID 6581 (Sound), unter die Lupe genommen und deren Steuerung von Basic aus aufgezeigt. Schließlich betrachten wir noch die beiden Ein-/Ausgabechips CIA 1 und CIA 2, denn auch mit diesen Bausteinen läßt sich in Basic einiges anstellen.

Im allgemeinen geschieht die Steuerung der drei gerade erwähnten Chips durch Register, die in bestimmten Speicherbereichen des C 64 adressiert sind. Die direkte Speichermanipulation ist bekanntlich mit den beiden Basic-Befehlen »PEEK« und »POKE« möglich. So lassen sich auch die VIC-, SID- und CIA-Register ansprechen und verändern.

In den Action-Grafikspielen, Zeichen- und Soundprogrammen passiert im Prinzip dasselbe, nur in der sehr viel schnelleren Maschinensprache.

Der VIC ist derjenige Chip, der für das Bild auf Ihrem Monitor oder Fernseher sorgt, während Sie mit dem C 64 arbeiten. Er ist dafür verantwortlich, daß die Buchstaben und Zeichen ordnungsgemäß, leserlich und in der richtigen Farbe auf dem Bildschirm erscheinen. Auch die Verarbeitung der HiRes-Grafik mit 320*200 Punkten Auflösung und die Steuerung der vielgerühmten Sprites stehen in seinem Aufgabenbereich.

Zu diesem Zweck besitzt der VIC 47 Register, die im Speicher des C 64 ab $D000 (dezimal 53248) liegen. Welches Register für welche Funktionen zuständig ist, ist in Bild 1 zu sehen.

Durch Setzen oder Löschen der jeweiligen Bits in den Registern kann die gewünschte Funktion ein- oder ausgeschaltet werden. So läßt sich der VIC auf verschiedene Darstellungsarten umschalten.

Der wohl bekannteste Zustand ist der Modus der normalen Zeichendarstellung, den der VIC nach dem Einschalten des Computers annimmt. In dieser Betriebsart holt sich der VIC der Reihe nach Bytes aus dem Video-RAM (auch Bildschirmspeicher genannt), und bringt dann das dazugehörige Bitmuster aus dem Charakter-ROM als entsprechendes Zeichen auf den Bildschirm.

Sämtliche 512 Zeichen, die der C64 kennt, sind im Charakter-ROM als Bitmuster gespeichert und werden auf die eben beschriebene Weise auf dem Bildschirm sichtbar gemacht. Dabei ist es prinzipiell möglich, den Zeichensatz nach eigenem Belieben zu verändern. Um dies von Basic aus zu tun, ist einiger Programmieraufwand nötig, der im Rahmen dieses Kurses nicht erklärt werden soll.

Die Farbinformation für jedes Zeichen auf dem Monitor entnimmt der VIC aus einem extra eingerichteten Farb-RAM, in dem jeweils 4 Bit pro Bildschirmposition zur Festlegung der Zeichenfarbe vorgesehen sind.

Etwas komplizierter ist die Darstellung von sogenannten Multicolor-Zeichen. Hierbei kann jedes Zeichen bis zu vier Farben haben. Man erreicht dies, indem man das Bit 4 des VIC-Registers 22 setzt.

Der VIC kontrolliert in diesem Modus zuerst die entsprechenden 4 Farbbit eines Zeichens und untersucht das erste Bit. Ist es 0, wird das Zeichen »normal« in der Größe von 8*8 Punkten dargestellt. Sollte das Bit gleich 1 sein, so wird das Zeichen nur noch in einer 4*8-Punktematrix mit doppelt breiten Punkten behandelt. 2 Bit im Bitmuster des Zeichens entsprechen also jetzt einem Bildschirmpunkt. Diese beiden Bits geben nun dem Grafik-Chip an, wo er den Farbwert des doppelt breiten Punktes entnehmen soll:

00 Hintergrund-Farbregister 0 (Register 33)
01 Hintergrund-Farbregister 1 (Register 34)
10 Hintergrund-Farbregister 2 (Register 35)
11 restliche 3 Bit aus dem Farb-RAM

Hochauflösende Grafik

Setzt man im VIC-Register 17 das Bit mit der Nummer 5, so arbeitet der Video-Chip von nun an im Bitmap-Modus, das heißt jedes gesetzte Bit im Speicher entspricht einem Punkt auf dem Bildschirm. Der VIC verarbeitet eine Auflösung von 320*200 Punkten, wofür er einen Grafikspeicher von 8 KByte benötigt. Der normale Bildschirmspeicher wird dabei zum Farb-RAM »umfunktioniert« und liefert jetzt die Informationen für die Farbe der gesetzten und nichtgesetzten Bildschirmpunkte.

Jedes Byte enthält hierbei den Farbwert für eine 8*8-Matrix von Punkten. Die ersten 4 Bit bestimmen die Farbe der gesetzten Punkte, die letzten 4 Bit sind für die nicht gesetzten Punkte der Matrix zuständig. Eine recht kompliziert klingende Sache also, die, wie wir noch sehen werden, auch nicht gerade einfach zu programmieren ist.

Wem die zwei Farben einer 8*8-Matrix nicht genug sind, der mag den Multicolor-Modus benutzen, der durch Bit 5 im Register 17 und durch Bit 4 des Registers 22 aktiviert wird. Er ermöglicht eine hochauflösende Grafik in vier verschiedenen Farben gleichzeitig. Die Auflösung beträgt hier aber nur noch die Hälfte des normalen HiRes-Modus, nämlich 160*200 Punkte. Jeder Bildschirmpunkt ist nun doppelt so breit und wird durch je 2 Bit im Grafikspeicher repräsentiert. Wie im Multicolor-Zeichenmodus gibt dieses Bitpaar die Farbe an, die der Grafikpunkt bekommen soll. Die Zustände der beiden Bits lassen den VIC wissen, woher er den Farbwert holen soll:

00 aus Hintergrund-Farbregister 0 (Register 33)
01 höherwertiges Halbbyte aus Video-RAM
10 niederwertiges Halbbyte aus Video-RAM
11 aus Farb-RAM

Auf diese Weise lassen sich Grafikbilder mit bis zu vier Farben nebeneinander erzeugen.

Was wäre ein anständiger VIC 6569 ohne seine Sprites, von denen er acht gleichzeitig steuern kann. Durch die Manipulation von bestimmten VIC-Registern ist es möglich, diese Sprites ziemlich schnell über den Bildschirm zu bewegen.

Sprites gibt's auch

Weitere Register zeigen Zusammenstöße mit Hintergrundzeichen oder anderen Sprites an, weshalb die Sprites hervorragend für Spiele geeignet sind, in denen Figuren schnell bewegt werden müssen.

Wie bei der hochauflösenden Grafik gibt es auch zwei verschiedene Spritearten: die zweifarbigen HiRes-Sprites und die Multicolor-Sprites, die aus vier Farben bestehen dürfen. So ist auch die Auflösung der Multicolor-Sprites nur halb so groß wie die der Zwei-Farben-Sprites, nämlich nur 12*21 Punkte statt 24*21.

Doch wie, so wird mancher Basic-Programmierer fragen, kann man diese so kompliziert klingenden Dinge selbst programmieren?

Um diese Frage zu beantworten, beginnt man am besten mit dem Einfachsten, nämlich mit der Manipulierung von Zeichen und deren Farben.

Was Sie sicherlich schon oft in Ihren Programmen getan haben, ist die Änderung der Rahmen- und Hintergrundfarben. Mit »POKE 53280,1« und »POKE 53281,1« wird der Bildschirm beispielsweise vollkommen weiß, während die Zeichenfarbe gleich bleibt. Diese Befehle bewirken nichts anderes, als daß die VIC-Register 32 (Rahmenfarbe) und 33 (Hintergrundfarbe 0) jeweils mit dem Farbwert Eins belegt werden (Eins entspricht der Farbe Weiß). Genauso wird auch bei den anderen Registern verfahren. Die Adressen der einzelnen Register sind in Bild 1 dezimal und hexadezimal aufgeführt.

Die Zeichenfarbe selbst kann man, wie aus dem C64-Handbuch ersichtlich, über die Tasten »1« bis »8« in Verbindung mit der CTRL- und der Commodore-Taste verändern. Die zweite Möglichkeit ist, die Farbwerte im Farb-RAM zu verändern. Im Normalfall befindet sich das Farb-RAM im Textmodus im Bereich von 55296 bis 56295 ($D800-$DBE7). »POKE 55299,1« bewirkt zum Beispiel, daß das vierte Zeichen in der obersten Zeile weiß wird. Ebenso kann man es mit allen Zeichen machen, wie das Beispielprogramm in Listing 1 zeigt.

Nun aber zur interessanteren Seite des Video-Prozessors, der hochauflösenden Grafik. Sie ist leider auch am kompliziertesten zu programmieren. Schon einfache Dinge, wie das Setzen oder Löschen eines Grafikpunktes, können beträchtliche Schwierigkeiten bereiten. Schuld daran ist die unge-

Register	Beschreibung	Adresse
0	X-Koordinate von Sprite 0	53248
1	Y-Kooridnate von Sprite 0	53249
2-15	X- und Y-Koordinaten der weiteren Sprites 1 bis 7	53250-53263
16	Überlaufregister für alle X-Koordinanten der Sprites. Bit 0 für Sprite 0, Bit 1 für Sprite 1 und so weiter	53264
17	Steuerregister 1 Bit 4 0= Bildschirm aus Bit 5 1= HiRes-Modus (hochauflösende Grafik) Bit 6 1= Extended Color	53265
18-20	zuständig für Rasterzeilen-Interrupt in Maschinensprache	53266-53268
21	Sprites an/aus. Jedem Sprite ist ein Bit zugeordnet 1= Sprite an 0= Sprite aus	53269
22	Steuerregister 2 Bit 4 1= Multicolor-Modus	53270
23	Sprite-Vergrößerung in Y-Richtung. Jedem Sprite ist ein Bit zugeordnet 1= Sprite wird in Y-Richtung vergrößert 0= Sprite wird normal dargestellt	53271
24	Basisadressen von Zeichengenerator und Video-RAM Bit 1-3 für Zeichengenerator Bit 4-7 für Video-RAM	53272
25-26	Interrupt Request- und Interrupt Mask-Registere	53273-53274
27	Hintergrund/Sprite Priorität Jedem Sprite ist ein Bit zugeordnet 1= Priorität der Hintergrundzeichen 0= Priorität des Sprites	53275
28	Sprite Multicolor/Normal. Jedem Sprite ein Bit zugeordnet. 1= Sprite ist Multicolor-Sprite 0= Sprite ist normal	53276
29	Sprite-Vergrößerung in X-Richtung Jedem Sprite ist ein Bit zugeordnet 1= Sprite wird in X-Richtung vergrößert	53277
30	Sprite-Sprite-Kollision. Jedem Sprite ist ein Bit zugeordnet Bei einer Kollision zweier Sprites sind die entsprechenden Bits gesetzt	53278
31	Sprite-Hintergrund-Kollision. Jedem Sprite ist ein Bit zugeordnet Bei der Kollision eines Sprites mit dem Hintergrund ist das entsprechende Bit gesetzt	53279
32	Rahmenfarbe	53280
33-36	Hintergrundfarbe 0-3	53281-53284
37-38	Sprite Multicolor-Register 1 und 2	53285-53286
39-46	Spritefarbe für Sprite 0 - Sprite 7	53287-53294

Bild 1. Die Register des VIC

AND- und OR-Wahrheitstabellen

AND

AND	0	1
0	0	0
1	0	1

Beispiele:

```
     11001100
AND  00001001
     --------
     00001000
```

OR

OR	0	1
0	0	1
1	1	1

```
     10011001
OR   10100100
     --------
     10111101
```

Bild 2. AND- und OR-Verknüpfungstafeln und Beispiele

wöhnliche Aufteilung des 8 KByte großen Grafikspeichers. Sie macht es erst durch einigen Programmieraufwand möglich, den Bildschirm in ein Koordinatensystem aufzuteilen und die einzelnen Punkte über die X- und Y-Koordinaten anzusprechen.

Im Normalfall, das heißt nach dem Einschalten des Computers, liegt der Grafikspeicher des VIC im Bereich von $2000 bis $3FFF (8192-16383) und das dazugehörige Farb-RAM an der Stelle des »normalen« Bildschirmspeichers, also von $0400 bis $07FF (1024-2047). Wie man sieht, liegt die Grafikseite direkt im Basic-Speicher und kann bei größeren Basic-Programmen zu Konflikten mit der Grafik führen. Es ist jedoch möglich, den Aktionsbereich unseres VIC im Gesamtspeicher zu verschieben und somit das Problem zu beseitigen. Wie man das macht, wird am Ende dieses Abschnitts beschrieben, da dies mit der eigentlichen Programmierung der Grafik nichts zu tun hat.

Bevor man eine Grafik erzeugen will, muß natürlich der VIC darüber informiert werden, daß er eine Grafik erzeugen und in welchem Modus dies geschehen soll.

Untersucht man die Tabelle in Bild 1, so entdeckt man in Register 17 und 22 die entprechenden Bits zum Ein- und Ausschalten des HiRes- und des Multicolor-Modus.

Doch wie, wird sich mancher Basic-Programmmierer fragen, kann man von Basic aus einzelne Bits löschen oder setzen? Um diese Frage zu beantworten, muß man etwas in die Welt der Binär-Arithmetik mit ihren logischen Verknüpfungen eintauchen:

Das Basic des C 64 beherrscht unter anderem auch zwei logische Operatoren, nämlich »AND« und »OR«. Sie bewirken eine logische Und- beziehungsweise Oder-Verknüpfung zwischen zwei Werten. Bild 2 zeigt eine Wahrheitstabelle, die angibt, wie sich die Verknüpfung auswirkt.

Etwas zur »Wahrheit«

Dieses Verknüpfen der einzelnen Bits kann auch zum gezielten Setzen oder Löschen von bestimmten Bits verwendet werden. Dazu verknüpft man einfach das entsprechende Byte mit einem bestimmten Wert, so daß das gewünschte Bit 1 oder 0 wird. Hier einige Beispiele:

Wir haben ein Byte mit dem Wert 155. Binär umgeformt sähe der Wert so aus:

10011011 = 155 (dezimal)

Angenommen in diesem Byte wollen wir das vierte Bit von rechts (also Bit 3) löschen, das heißt gleich Null setzen. Nun braucht man einen Wert, der mit »AND« oder »OR« verknüpft das Bit Nummer 3 löscht, die restlichen Bits aber unverändert läßt.

Betrachtet man die Wahrheitstabellen für »AND« und »OR« (Bild 2), so stellt man fest, daß sich die Und-Verknüpfung dafür eignen würde.

```
     10011011 = 155
AND  11110111 = 247
     --------------
     10010011 = 147
```

Und schon ist das besagte Bit gelöscht. Allgemein erreicht man den dezimalen Wert zur Und-Verknüpfung durch folgende Formel:

255-(dezimaler Wert des zu löschenden Bits)

In unserem Beispiel ist der Stellenwert unseres Bits Nummer 3 gleich Drei. Der Dezimalwert errechnet sich also aus 2^3, das ergibt 8. Setzen wir ihn in die Formel ein, so erhalten wir den gewünschten Wert 247 zur Und-Verknüpfung:

155 AND (255-8) = 147

Nach demselben Muster läßt sich selbstverständlich auch ein Bit setzen. Hier allerdings bietet sich der »OR«-Operator, also die Oder-Verknüpfung an. Bleiben wir beim Beispiel von vorhin, nur daß nun Bit Nummer 6 (das zweite Bit von links) auf Eins gesetzt werden soll:

```
     10011011 = 155
OR   01000000 =  64
     --------------
     11011011 = 219
```

Das entsprechende Byte muß nur mit dem dezimalen Wert des zu setzenden Bits ge-oder-t werden, um dieses Bit gleich Eins werden zu lassen. Der Stellenwert des Bits Nummer 6 beträgt 2^6, und somit nach Adam Riese 64:

155 OR 64 = 219

Nach diesem Prinzip kann man sämtliche Bits in einem Byte setzen oder löschen. Genauso werden auch die einzelnen Bits in den Registern des VIC 6569 manipuliert.

Zum Einschalten der hochauflösenden Grafik sind einige POKE-Befehle nötig. Im folgenden einige Basic-Zeilen, die die hochauflösende Grafik einschalten:

```
60000 REM ***GRAFIK EINSCHALTEN ***
60100 REM
60200 VIC=53248:REM STARTADRESSE VIC
60300 POKE VIC+17, PEEK(VIC+17) OR 2↑5
60400 POKE VIC+17, PEEK(VIC+17) OR 2↑6
60450 POKE VIC+24, PEEK(VIC+24) OR 8
```

Nun muß man noch entscheiden, ob man im vierfarbigen Multicolor-Modus mit einer niedrigeren Auflösung arbeiten will oder die Zweifarben-Grafik mit 320 * 200 ansprechbaren Punkten benutzen möchte.

Für die hochauflösende Grafik muß man im Normalzustand des Computers eigentlich nichts mehr verändern, sollte jedoch sicherheitshalber das Multicolor-Bit in Register 22 löschen. Das geschieht mit:

```
60500 POKE VIC+22,PEEK(VIC+22) AND 255-16
```

Soll der Multicolor-Modus eingeschaltet werden, ist ein entsprechender POKE-Befehl zu verwenden, der das Bit setzt:

```
60500 POKE VIC+22,PEEK(VIC+22) OR 16
```

Doch damit ist das Einschalten der Grafik genaugenommen noch nicht zu Ende. Startet man obiges Programm, so schaltet der VIC zwar auf Grafik um, doch man wird nur ein buntes Durcheinander von verschiedenfarbigen Punkten auf dem Bildschirm erkennen.

Grafik einschalten

Das liegt daran, daß der jetzt vom VIC als Grafikspeicher deklarierte Bereich (normalerweise $2000-$3FFF) zuvor anderweitig benutzt wurde. Daher ist er mit »wilden« Werten »gespickt«, die der Video-Chip in seinem Bitmap-Modus als wirres Muster auf den Bildschirm bringt.

Aus diesem Grund muß der Grafikspeicher vor Benutzung stets »leergeräumt«, das heißt sämtliche 64000 Bit gelöscht werden. Das ist mit einer einfachen FOR-NEXT-Schleife zu bewerkstelligen. Im folgenden Basic-Programm ist dies realisiert.

```
61000 REM ***GRAFIKSPEICHER LÖSCHEN ***
61100 REM
61200 VD= 8192:REM STARTADRESSE GRAFIKSPEICHER
61300 FOR X=VD to VD+8000
61400 POKE X,0
61500 NEXT X
```

Fügt man diese Basic-Zeilen zu unserem Programm hinzu, kann man feststellen, daß das »Punktewirrwarr« langsam aber sicher verschwindet. Die Farben bleiben jedoch, wo sie sind. Wir erinnern uns, daß im Grafik-Modus die Farbe jedes 8 * 8-Kästchens in einem gesonderten Farb-RAM festgelegt ist. In unserem Fall liegt es, wie schon erwähnt, im Bereich des normalen Textbildschirms.

Jedem 8 * 8-Kästchen im Grafikspeicher ist ein Byte im Farb-RAM zugeordnet. Die vier höchstwertigen Bits eines jeden Bytes geben dabei die Farbe der gesetzten, die vier niederwertigen Bits die der nichtgesetzten Punkte an.

Um ein »anständiges« Grafikbild zu erzeugen, muß die Farbe jedes Kästchens einheitlich gesetzt werden, was wiederum in einer FOR-NEXT-Schleife geschehen kann:

```
62000 REM ***FARBE SETZEN ***
62100 REM
62200 FR=1024:REM ANFANGSADRESSE FARBRAM
62300 FARBE= 0*16 + 14:REM PUNKTE SCHWARZ-
      HINTERGRUND HELLBLAU
62400 FOR X=FR TO FR+1000
62500 POKE X,FARBE:REM FARBWERT SETZEN
62600 NEXT X
```

Jetzt haben wir einen leeren Grafik-Bildschirm mit blauem Hintergrund. Die Grafikpunkte werden später in der Farbe Schwarz auf dem Monitor erscheinen.

Drücken Sie nun RUN/STOP und RESTORE zugleich, gelangen Sie wieder in den normalen Text-Modus zurück. Ein laufendes Basic-Programm wird dabei aber unterbrochen. Soll dies nicht geschehen, muß man die Grafik vom Programm aus wieder abschalten. Dazu müssen lediglich die Bits wieder gelöscht werden, mit denen man den Grafik-Modus zuvor eingeschaltet hatte.

Das sind also die Bits 5 und 6 im VIC-Register 17:

```
63000 REM ***GRAFIK AUSSCHALTEN ***
63100 REM
63200 VIC=53248:REM GRUNDADRESSE VIC
63300 POKE VIC+17, PEEK(VIC+17) AND 255-32:REM
      BIT 5
63400 POKE VIC+17, PEEK(VIC+17) AND 255-64:REM
      BIT 6
```

Damit hat man alle Programmteile zusammen, um die HiRes-Grafik zu bedienen.

Jetzt zum schwierigeren Teil, dem Setzen und Löschen von einzelnen Punkten. Hierbei müssen einige Berechnungen angestellt werden, bevor der Punkt auf dem Bildschirm erscheint.

Es ist leider nicht möglich, Punkte direkt über X- und Y-Koordinaten in den Grafikspeicher einzugeben. Das liegt am recht unhandlichen Aufbau des Grafikspeichers.

So wird das Grafikbild nicht Zeile für Zeile hintereinander im Speicher abgelegt, sondern wie im Zeichengenerator die einzelnen Zeichen in Blöcken von jeweils 8 * 8 Punkten.

8 Bytes des Grafikspeichers sind jeweils für eine solche 8 * 8-Matrix (auch Grafik-Matrix oder Grafik-Kästchen genannt) zuständig. Je eine 8-Punkte-Reihe in dieser Matrix wird durch ein Byte dargestellt.

Die nächsten 8 Byte liefern dann die Information für die zweite Matrix, die folgenden acht für den dritten Block und so weiter bis zum rechten Rand des Bildschirms. Dann beginnt die zweite Matrix-Zeile nach dem gleichen Schema und wird so Kästchenzeile für Kästchenzeile bis zum unteren Bildschirmrand fortgeführt. Bild 3 veranschaulicht dieses Prinzip.

Insgesamt passen 320/8, also gleich 40 der 8 * 8-Blöcke in eine Reihe und 200/8, also gleich 25 8 * 8-Blöcke in eine Spalte der Grafik. Man sieht, daß dies dem Text-Modus ähnelt, da ja ein normales Zeichen auch aus einer 8 * 8-Matrix besteht.

Wegen diesem, für den Programmierer etwas komplizierten Bildaufbau einer HiRes-Grafik, ist es nur mit einigen Berechnungen möglich, die einzelnen Punkte wie in einem Koordinatensystem nach X- und Y-Koordinaten festzulegen. Die X-Achse legen wir dabei waagrecht von links nach rechts und numerieren die Punkte in dieser Richtung nach ihrer Anzahl von 0-319. Die Y-Achse wird senkrecht von oben nach unten von 0-199 angelegt. Jeder Punkt wird dadurch mit seinem X- und Y-Wert in seiner Position auf dem Bildschirm eindeutig bestimmt. Um von den Koordinaten (X,Y) eines Punktes auf das richtige Byte und Bit im Grafikspeicher zu schließen, bedarf es einiger Überlegungen. Betrachten wir zunächst die Y-Koordinate.

Im ersten Schritt wird der Y-Wert durch acht dividiert. Dadurch ermittelt man die Kästchenzeile des zu berechnenden Grafikpunktes:

Kästchenzeile = INT(Y/8)

Wie schon ausgerechnet, passen 40 Grafik-Kästchen in jeweils eine der Kästchenzeilen. Jede Grafik-Matrix benötigt 8 Byte. Eine Kästchenzeile belegt demnach 40 * 8 = 320 Byte des Grafikspeichers.

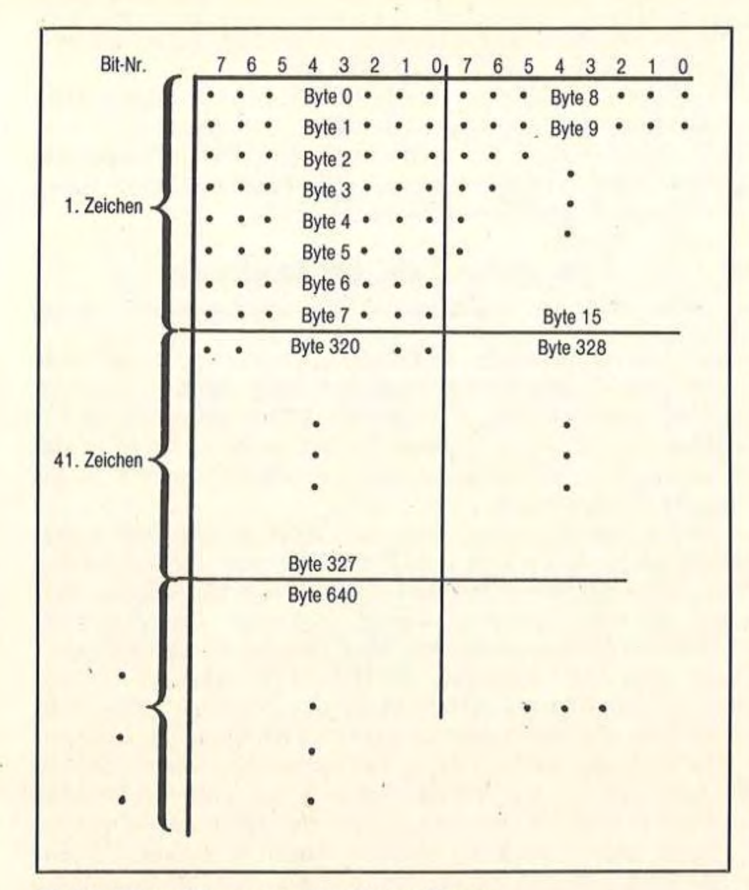


Bild 3. Bildschirmaufbau im HiRes-Modus

Multiplizieren wir nun das Ergebnis der oben beschriebenen Division mit 320, erhalten wir jetzt die relative Startadresse der errechneten Kästchenzeile im Grafikspeicher.
Adresse der Kästchenzeile = 320 * INT(Y/8)

Schließlich darf man die restlichen Bytes der obigen Division nicht vergessen und zählt sie zur besagten Adresse dazu.

Der Divisionsrest berechnet sich sehr einfach:
DivRest = Y-(Kästchenzeile*8)

Kennt man die Gesetze der Binärarithmetik, erkennt man sofort eine einfachere Berechnung des Restes einer Division durch den Wert 8:
DivRest = Y AND 7

Addieren wir den Rest zu unserer Adresse hinzu, erhalten wir letztendlich folgende Formel:
Y-Adresse = 320 * INT(Y/8) + (Y AND 7)

Jetzt benötigt man noch den X-Wert des Grafikpunktes. Wie oben dividiert man ihn durch 8, um analog dazu die Nummer der Kästchenspalte, in der sich der Punkt befindet, zu erhalten.
Kästchenspalte = INT(X/8)

Daraus läßt sich durch eine Multiplikation mit acht die Anzahl der Bytes berechnen, die zur vorhin ermittelten Y-Adresse dazugezählt werden müssen:
Bytes = 8 * INT(X/8)

Damit hätte man die Adresse des Grafik-Bytes, in dem der betreffende Punkt gespeichert ist, gefunden.
Grafikadresse = 320 * INT(Y/8) + (Y AND 7) + (8 * INT(X/8))

Was geschieht mit dem Rest der Division des X-Wertes; muß auch er eine Bedeutung haben? Er gibt an, welches Bit in der Adresse gesetzt werden muß, um unseren Grafikpunkt auf dem Bildschirm erscheinen zu lassen.

Man berechnet die Nummer des Bits folgendermaßen:
Bitnummer = 7-(X AND 7)

Nun kann der Grafikpunkt an der richtigen Stelle gesetzt oder gelöscht werden. Wie man einzelne Bits von Basic aus setzt oder löscht, ist Ihnen ja bereits bekannt.

Listing 2 und 3 zeigen zwei kleine Basic-Programme zum Setzen und Löschen von Grafikpunkten, die unsere erstellten Formeln verwenden.

Durch weitere Berechnungen lassen sich daraus auch weitere Grafikfunktionen programmieren. Probieren Sie doch einmal ein Basic-Programm zu schreiben, das eine Linie zwischen zwei vorgegebenen Punkten (X1;Y1) und (X2;Y2) zieht.

Im Multicolor-Modus (Bit 4 = 1 im VIC-Register 22) läuft die Punkteberechnung ähnlich ab. Der entscheidende Unterschied ist die nur halbe Auflösung von 160, dafür aber doppelt breiten Punkten in X-Richtung. In Y-Richtung hat sich nichts geändert. Die Formel zur Berechnung der Y-Adresse des Grafikpunktes ist demnach im HiRes-und im Multicolor-Modus identisch (siehe auch Bild 4).

Man erkennt, daß nun 2 Bit für einen Grafikpunkt zuständig sind. Ein Byte hat also nur noch für vier Punkte Platz. Das bedeutet eine Änderung bei den Formeln »Bytes« und »Bitnummer«. Auch das Setzen oder Löschen von Punkten ist nun etwas komplizierter.
Bytes = 8 * INT(x/4)
Bitnummer1 = (3-(X-(4*INT(X/4))))*2
Bitnummer2 = Bitnummer1 + 1

Da jetzt auch 2 Bit pro Punkt verändert werden müssen, gibt es auch zwei Bitnummern. Listing 4 zeigt das Setzen eines Punktes in Multicolor.

Als Anwendungsbeispiele sollen die Listings 5 und 6 dienen, die beide eine Sinuskurve auf den Bildschirm bringen. Das Programm in Listing 5 tut dies im HiRes-Modus; das Programm in Listing 6 verwendet die Multicolor-Grafik.

Sprites richtig programmiert

Die Fähigkeit des VIC, Sprites zu erzeugen, ist wohl jedem bekannt. Sprites sind genau gesagt kleine Grafiken von 24*21 Punkten Größe, die frei über den Bildschirm bewegt werden können. Der VIC kann 8 Sprites gleichzeitig auf dem Bildschirm verwalten.

Er tut dies über einige seiner Register, die sämtliche Informationen über alle Sprites enthalten.

Anders als bei der hochauflösenden Grafik wird ein Sprite Punkt für Punkt, das heißt Bit für Bit der Reihe nach im Speicher abgelegt (siehe Bild 5).

So ist die Erstellung von Sprites etwas einfacher als die Erzeugung von Grafik.

Jedes Sprite besteht aus 24*21, also 504 Punkten und braucht somit 504/8 = 63 Byte Speicherplatz.

Die Farbinformationen für »normale« Sprites entnimmt der VIC aus eigens dafür angelegten Sprite-Farbregistern (VIC-Register 39-46; Adressen 53287 bis 53294). Eine Farbänderung von beispielsweise Sprite Nummer 3 auf rot ist mit POKE 53290,2 zu bewerkstelligen.

Damit liegt der Nachteil »normaler« Sprites schon auf der Hand. Sie können nur aus jeweils einer Farbe bestehen, die in das entsprechende Farbregister gePOKEt wird.

Wenn es normale Sprites gibt, so muß es wohl auch noch andersartige Sprites geben. Die gibt es auch: es sind die Multicolor-Sprites.

Wie bei der Multicolor-Grafik kann ein solches Sprite aus bis zu vier verschiedenen Farben bestehen (wenn man die Hintergrundfarbe mitrechnet). Doch wie sollte es anders sein, ein solch buntes Sprite besitzt nur noch die halbe Auflösung von 12*21 Punkten. Auch hier sind die Punkte wieder doppelt so breit wie normal und werden jeweils durch 2 Bit im Speicher repräsentiert. Dabei bestimmt wieder der Zustand der beiden Bits, woher der Farbwert für diesen Punkt stammen soll:

00 Hintergrund-Farbregister
01 Multicolor-Register 0 (Register 37)
10 Multicolor-Register 1 (Register 38)
11 Sprite-Farbregister (Register 39-46)

Man sieht, daß alle Multicolor-Sprites zwei gemeinsame Farbregister haben. Das sind die Multicolor-Register 0 und 1. Nur das Register für die vierte Farbe ist für jedes Sprite extra angelegt, und kann somit für alle Sprites unterschiedlich gewählt werden. Der Aufbau entspricht dem eines normalen Sprites mit dem einen Unterschied, daß nun 2 Bit einen Punkt darstellen (Bild 6).

Ob ein Sprite »normal« oder in Multicolor dargestellt werden soll, ist in einem besonderen VIC-Register festgehalten (Register 28; Adresse 53276). Jedem Sprite ist ein Bit dieses Registers zugeordnet. Ein gesetztes Bit bedeutet für den Video-Chip, daß es sich bei dem entsprechenden Sprite um ein Multicolor-Sprite handelt. POKE 53276,128 macht zum Beispiel das Sprite Nummer 7 zum Multicolor-Sprite.

Nachdem man sich überlegt hat, welche Sprites »normal« oder im Multicolor-Modus verwaltet werden sollen, stellt sich noch die Frage, wie man seinen Spriteentwurf am einfachsten in den Speicher bringt.

Hat man keinen Sprite-Editor zur Hand, muß man diese Arbeit wohl oder übel selbst tun. Dazu legt man sich auf Papier ein Raster von 24*21 Punkten an und zeichnet sich seinen Entwurf Punkt für Punkt hinein. Auf einem zweiten Entwurf zeichnet man sich sein Sprite im Bitmuster auf. Jeder gesetzte Punkt entspricht einer Eins im Muster, alle ungesetzten Punkte haben den Wert Null (Bild 7).

Für ein Multicolor-Sprite legt man entsprechend ein 12*21-Raster an und verfährt nach demselben Muster. Der Unterschied ist nur, daß man im zweiten Entwurf nun pro Punkt 2 Bit entsprechend der Farbe einträgt, die der Punkt später haben soll.

Anschließend rechnet man der Reihe nach 8 Bit (also immer ein Byte) in einen dezimalen Wert um und notiert sie neben dem Raster (Bild 7).

Hat man seine 63 Byte zusammengestellt, müssen sie durch POKE-Befehle an den richtigen Platz im Speicher gebracht werden. Dazu muß man sie in den 16-KByte-Adressierungsbereich des VIC legen, damit sie auch gelesen und verarbeitet werden können. Zu diesem Zweck wird der VIC-Bereich in Blöcke zu je 64 Byte eingeteilt. Jeder Block erhält eine Nummer von 0 bis 255. Unsere Spritedefinition muß nun in einem dieser Blöcke untergebracht werden.

In der Einschaltkonfiguration gibt es allerdings einige Einschränkungen. Hier überschneidet der Adreßbereich des VIC einige wichtige Teile des Speichers, wie zum Beispiel die Zeropage.

Block Nummer 0 hätte die Startadresse $0000, Block 1 die Adresse $0040 (dezimal 64) und so weiter.

Setzen wir unser Sprite in diesen Bereich, hätte das verheerende Folgen für die Funktionsbereitschaft unseres C 64. Die ersten Blöcke, die keinen so sehr wichtigen Speicher belegen, sind die Blöcke

11 Bereich $02C0-02FE (704-766)
13 Bereich $0340-037E (832-894)
14 Bereich $0380-03BE (896-958)
15 Bereich $03C0-03FE (960-1022)

Arbeiten Sie mit der Datasette, kann es dabei auch zu Problemen kommen, da sich die Blöcke 13, 14 und 15 im Kassettenpuffer befinden.

Danach ist es erst wieder ab Block 128, das heißt ab Adresse $2000 (8192) möglich, gefahrlos Sprite-Definitionen abzulegen. Vorsicht sei aber geboten bei großen Basic-Programmen oder wenn Sie gleichzeitig mit HiRes-Grafik arbeiten, da ja bekanntlich ab $2000 (8192) das Grafikbild abgelegt ist. Verschieben Sie den Adreßbereich des VIC, können Sie diesen Problemen teilweise aus dem Weg gehen.

Wollen wir aber nur eine Spritedefinition abspeichern, genügen uns die Blöcke 11 und 13 bis 15.

Nehmen wir für unser Sprite den Block 11 und schreiben eine kleine Routine, die uns alle 63 Byte der Definition in diesen Block POKEt. Am besten legt man die Spritewerte in DATA-Zeilen ab und benutzt eine FOR-NEXT-Schleife (Listing 7 verwendet ein kleines Demo-Sprite).

So stehen sie im Speicher

Damit ist die Arbeit der Definition noch nicht zu Ende. Jetzt muß dem VIC mitgeteilt werden, daß unser Sprite ab Block 11 gespeichert ist. Dazu benutzt der VIC je einen Zeiger für jedes der 8 Sprites. Er zeigt auf den Block, aus dem er die Definition für das augenblickliche Aussehen eines jeden Sprites nehmen soll.

Die Zeiger liegen am Ende des Video-RAMs. Das Video-RAM ist genau 1 KByte, also 1024 Byte groß. Zur Darstellung des Bildschirms werden aber nur 40*25 = 1000 Byte benötigt. Der Rest, das sind 24 Byte, liegt brach. Davon werden die letzten 8 Byte als Sprite-Zeiger benutzt. Das sind in unserem Fall die Adressen $07F8-07FF (dezimal 2040-2047). Die Adresse 2040 enthält den Zeiger für Sprite Nummer null, die Speicherstelle 2041 ist für Sprite 2 zuständig und so weiter bis zum Zeiger für Sprite 7 bei Adresse 2047.

Verlegen wir das Video-RAM in einen anderen Bereich (diese Möglichkeit besteht), liegen die Sprite-Zeiger selbstverständlich in anderen Speicherstellen. In diesen Zeigern stehen nun die Blocknummern, bei denen die Definitionen für jedes der 8 Sprites stehen.

Wollen wir dem Sprite 0 zum Beispiel das Aussehen unseres oben genannten Demo-Sprites geben, stellen wir den Zeiger für Sprite 0 auf Block 11 (in dem unsere Sprite-Definition gespeichert ist). Mit POKE 2040,11 ist das schnell geschehen. Soll ein zweites Sprite genauso aussehen, genügt ein entsprechender POKE-Befehl: POKE 2042,11.

Wir legen den gewünschten Zeiger auf denselben Block, und somit sieht Sprite Nummer 2 so aus wie Sprite Nummer 0. Man kann also jede beliebige Sprite-Definition für jedes beliebige Sprite nutzen. Eine Multicolor-Definition für ein normales Sprite zu verwenden, ist natürlich nicht ratsam. Sie können es ja zum Spaß einmal ausprobieren.

Es ist möglich, jederzeit die Definition eines Sprites zu wechseln, indem man den Zeiger auf einen anderen Block stellt. So kann man das Aussehen eines Sprites blitzschnell ändern und somit den Effekt einer Animation erzielen.

POKE 2040,13 ändert das Aussehen des Sprites 0 sofort auf die Definition in Block 13.

Bevor man jedoch ein Sprite auf dem Bildschirm sehen kann, muß es erst eingeschaltet werden. Dazu besitzt der VIC das Register 21 mit der Adresse 53269, das den Einschaltzustand aller Sprites enthält. Für jedes Sprite ist wieder ein Bit des Registers zuständig. Schalten wir also unser Sprite 0 ein:

POKE 53269,1 (Bit 0 wird gesetzt)

Jetzt ist möglicherweise immer noch nichts auf dem Bildschirm zu entdecken, das unserem Sprite gleicht. Womöglich steht es in einer Position außerhalb des sichtbaren Bildschirmbereichs.

Der Bereich, in dem sich ein Sprite befinden kann, ist wie ein Koordinatengitter in X- und Y-Richtung aufgeteilt. Das Gitter ist aber größer als der sichtbare Bildschirm. Es ist möglich, das Sprite einzelpunktweise von 0-511 in X-Richtung und von 0-255 in Y-Richtung zu bewegen.

Der Nullpunkt unseres Sprite-Koordinatengitters liegt weit außerhalb des Bildschirms in der linken oberen Ecke. Positionieren wir das Sprite auf die Koordinaten 20,30, ist es vollständig in der linken oberen Ecke sichtbar. Die Übergröße

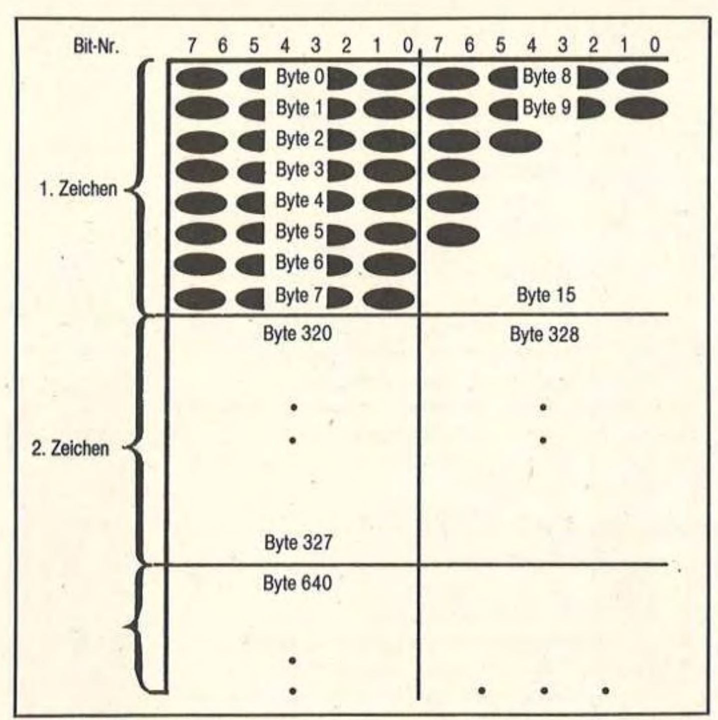


Bild 4. So werden die Bildschirmpunkte im Multicolor-Modus beschrieben

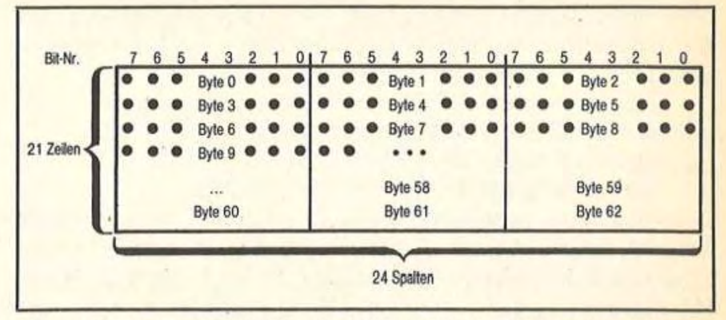


Bild 5. Aufbau eines HiRes-Sprites mit 24*21 Punkten. Das ist die maximal erreichbare Auflösung.

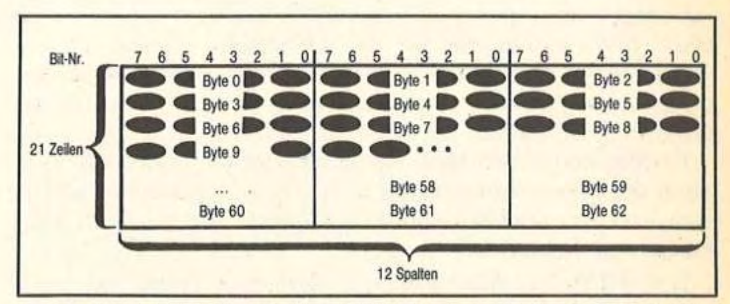


Bild 6. Ein Multicolor-Sprite verfügt nur mehr über eine Auflösung von 12*21 Punkten

des Sprite-Bewegungsraumes führt zu dem recht schönen Effekt, daß man das Sprite zu allen Seiten des Bildschirms punktweise hinauswandern lassen kann. Doch wo werden die X,Y-Positionen für jedes Sprite gespeichert?

Bewegung von Sprites

Betrachtet man Bild 1, die Registertabelle des VIC, entdeckt man sie sofort. Die ersten 17 Register sind für die Position der 8 Sprites reserviert. Durch einfache POKE-Befehle kann man die X- und Y-Werte aller Sprites schnell verändern und somit jedes Sprite blitzartig an sämtliche Positionen des Sprite-Koordinatengitters versetzen:

```
POKE 53248,150
POKE 53249,111
```

Mit diesen Befehlen wird zum Beispiel Sprite 0 auf Position x=150 und y=111 plaziert.

Die X-und Y-Register aller Sprites können nur Werte von 0-255 annehmen. Will man ein Sprite auf eine X-Position verschieben, die größer als 255 ist (bis maximal 511), muß ein Übertrag auf ein anderes Register geschehen. Dabei muß im Register 16 (53264), dem Übertragsregister, das Bit für das betreffende Sprite gesetzt werden.

In Listing 8 wurde das eben Beschriebene in ein Programm umgesetzt, das die Bewegung eines Sprites verdeutlichen soll.

Doch damit sind die Fähigkeiten unseres Videoprozessors noch nicht ausgeschöpft. So kann jedes Sprite seine Größe in X- und Y-Richtung verdoppeln.

Für diesen Effekt sind die Register 23 (Adresse 53271) für die Y-Vergrößerung und 29 (Adresse 53277) für die X-Vergrößerung verantwortlich. Wieder ist in diesen Registern für jedes Sprite ein Bit zuständig, dessen Zustand dem VIC mitteilt, ob das Sprite normal oder vergrößert dargestellt werden soll. POKE 53271,64 beispielsweise vergrößert das Sprite Nummer 6 in Y-Richtung auf doppelte Höhe.

Betrachtet man die Registertabelle in Bild 1, entdeckt man noch einige andere Sprite-Register, mit deren Hilfe man weitere Eigenschaften der Sprites einstellen kann.

Register 27 (Adresse 53275) erlaubt für jedes Sprite die Wahl, ob es auf dem Bildschirm vor den Hintergrundzeichen oder dahinter bewegt werden soll. Ist das entsprechende Bit gesetzt, so erscheint das angesprochene Sprite hinter den Hintergrundzeichen. Ein gelöschtes Bit gibt dem Sprite Priorität vor dem Hintergrund.

Hintergrundzeichen sind in diesem Fall alle gesetzten Punkt die sich außer den Sprites noch auf dem Bildschirm befinden. Das können wahlweise Zeichen oder auch Grafikpunkte sein.

Stoßen 2 Sprites zusammen, so wird das von Register 30 (Adresse 53278) angezeigt. Der VIC setzt dabei die entsprechenden Bits der Sprites, die kollidiert sind. Sind beispielsweise Bit 5 und 6 gleich 1, zeigt dies eine Berührung von Sprite Nummer 5 mit Sprite Nummer 6 an.

Dabei ist aber Vorsicht geboten, da nach dem Zusammenstoß die Bits nicht mehr selbständig vom Videoprozessor zurückgesetzt werden. Nach jeder Kollisionsabfrage sollte daher das Register regelmäßig vom Basic-Programm gelöscht werden. Mit POKE 53278,0 ist das schnell getan.

Das nächste Register mit der Nummer 31 hat die Adresse 53279 und registriert eine Berührung eines Sprites mit einem Hintergrundzeichen. Wie in Register 30 besagt ein gesetztes Bit, daß das entsprechende Sprite gerade einen Punkt des Hintergrundes berührt hat. Auch hier muß das Register danach wieder von Hand gelöscht werden. So sollte der Befehl POKE 53279,0 nach der Kollisionsabfrage nicht fehlen.

Die Abfrage, ob bestimmte Bits gesetzt sind, ist mit einer IF-THEN-Abfrage leicht zu bewerkstelligen.

Wollen wir überprüfen, ob die Sprites 2 und 7 zusammengestoßen sind, so benutzen wir die folgende Basic-Zeile:

```
IF (PEEK(53278) AND (2↑2) <> 0) AND (PEEK(53278)
AND (2↑7) <> 0) THEN ...   :REM KOLLISION
```

Damit hätten wir sämtliche Funktionen des VIC beisammen, um alle 8 Sprites auf dem Bildschirm »tanzen« zu lassen.

Bleibt nur noch die Möglichkeit, den Adreßbereich des VIC-Chips zu verschieben.

In manchen Fällen, wie bei besonders langen Basic-Programmen, ist es manchmal notwendig, den HiRes-Grafikspeicher in einen anderen Bereich zu verlegen, als er nach dem Einschalten steht. Durch bestimmte Register im VIC und im CIA 2 (den wir auch noch besprechen werden) kann der 16-KByte-Adreßbereich des VIC im Speicher verschoben werden. Zuständig dafür sind die Bits 0 und 1 im

Register 0 der CIA 2, die je nach Zustand den Arbeitsbereich des VIC in bestimmte Speicherbereiche Ihres C64 legen:

Bits 1/0	Arbeitsbereich des VIC
11	$0000-$3FFF (0-16383)
01	$4000-$7FFF (16384-32767)
10	$8000-$BFFF (32768-49151)
00	$C000-$FFFF (49152-65535)

Verlegen Sie zum Beispiel den Arbeitsbereich des VIC auf $4000-7FFF (16384-32767), werden auch sämtliche Bildspeicherfunktionen mitverschoben. So liegt nun das Video-RAM im Bereich von $4400-47FF. Auch der Zeichengenerator liegt an anderer Stelle und der Beginn der Sprite-Definitionsblöcke beginnt dann bei $4000 (16384), das heißt Block 0 fängt bei Adresse 16384 und nicht mehr bei Adresse 0 an.

```
POKE 56576, PEEK(56576) AND (255-2)
```

würde das eben Erklärte realisieren. Ohne weitere Veränderungen wäre aber so kein anständiges Arbeiten mit dem Computer mehr möglich.

Innerhalb des 16-KByte-Adressierungsbereiches läßt sich auch der Bildschirmspeicher in 1-KByte-Schritten verschieben. Um das durchzuführen, sind die Bits 4 bis 7 im VIC-Register 24 zuständig.

Diese 4 Bit repräsentieren die dezimalen Werte von 0 bis 15 und geben an, in welchem 1-KByte-Block sich das Video-RAM befindet. Im Einschaltzustand haben die 4 Bit den Binärwert %0001, was dezimal dem Wert 1 entspricht (Bildschirm in Block 1 bei $0400; dezimal 1024). Wollen wir den Bildschirm nach $0800 (2048), also nach Block 2 verschieben, ist es notwendig, die 4 Bit mit dem Wert 2 zu belegen. Die Bits müssen dann wie folgt gesetzt werden: %0010, was dem Wert 2 entspricht.

Wie das Video-RAM kann auch der Zeichengenerator, das heißt der Speicher, an dem die Bitmuster der Zeichen abgelegt sind, innerhalb des VIC-Arbeitsbereiches verschoben werden. Dies ist lediglich in 2-KByte-Schritten möglich und wird mit den Bits 1 bis 3 des VIC-Registers 24 ermöglicht.

»Verschiebereien«

Auch hier repräsentieren diese 3 Bit einen Wert, der beschreibt, in welchem 2-KByte-Block der Zeichengenerator liegt. Im Normalfall sind die Bits folgendermaßen gesetzt: %010, was dezimal dem Wert 2 entspricht. Der Zeichengenerator liegt also im zweiten 2-KByte-Block des VIC-Arbeitsbereiches. Theoretisch wäre dies der Bereich von $1000-1FFF (dezimal 4096-8191). Eine hardwaremäßige Verschaltung hat aber den Effekt, daß beim Zugriff des VIC auf diesen Adreßbereich automatisch auf das Charakter-ROM im Bereich $D000-DFFF zugegriffen wird. Dort steht der normale Zeichensatz des C64 und bringt somit die Zeichen auf den Bildschirm.

Verschiebt man den Zeichengenerator beispielsweise auf den Bereich $2000-2FFF (dezimal 8192-12287; Bits 1 bis 3 auf %100), werden die Bitmuster für die Zeichen tatsächlich aus diesem Bereich geholt. Haben Sie dann dort keinen eigenen Zeichensatz gespeichert, wird sich ein heilloses Durcheinander auf dem Bildschirm einstellen. Diese Tatsache ermöglicht aber auch die Herstellung eigener Zeichensätze und kann somit von Vorteil sein.

Schließlich kann auch noch der Grafikspeicher innerhalb des 16-KByte-Arbeitsbereiches auf zwei Positionen verschoben werden. Position 0 wäre der untere 8-KByte-Bereich und Position 1 der beim Einschalten eingestellte obere 8-KByte-Bereich des VIC-Arbeitsspeichers. Liegt der VIC-Adreßbereich von $0000-3FFF (0-16383), so kann also wahlweise der Grafik-Bildschirm auf $0000-$1FFF oder auf $2000-3FFF gelegt werden. Ein Verlegen auf $0000 wäre

Register	Bedeutung	Adresse
0	Tonfrequenz für Stimme 1 höherwertiges Byte	54272
1	Tonfrequenz für Stimme 1 niederwertiges Byte	54273
2-3	Pulsbreite für Rechteckschwingung höher- und niederwertiges Byte	54274-54275
4	Steuerregister für Stimme 1 Bit 0 KEY: 1= Ton beginnt nach der entsprechenden Hüllkurve zu erklingen Bit 1 SYNC: 1= synchronisiert Oszillator 1 und 3 Bit 2 RING: 1= Dreiecksausgang wird durch eine Modulation von Oszillator 1 und 3 ersetzt Bit 4 TRI: 1= Dreiecksschwingung Bit 5 SAW: 1= Sägezahnschwingung Bit 6 PUL: 1= Rechteckschwingung (Pulsbreite in den Registern 2 und 3) Bit 7 NSE: 1= Rauschen	54276
5	Hüllkurvenregister 1 Bits 0-3 : Decay-Wert (0-15) Bits 4-7 : Attack-Wert (0-15)	54277
6	Hüllkurvenregister 2 Bits 0-3 : Release-Wert (0-15) Bits 4-7 : Sustain-Wert (0-15)	54278
7-13	Entsprechende Register für Stimme 2 Ausnahmen SYNC synchronisiert Oszillator 2 und 1 RING Dreiecksausgang wird durch Modulation von Oszillator 2 und 1 ersetzt	54279-54285
14-20	Entsprechende Register für Stimme 3 Ausnahmen SYNC synchronisiert Oszillator 3 und 2 RING Dreiecksausgang wird durch Modulation von Oszillator 3 und 2 ersetzt	54286-54292
21-22	Filterfrequenzregister nieder- und höherwertiges Byte Die Frequenz errechnet sich mit diesem Wert (FW) durch F=(30+FW*5.8)	54293-54294
23	Filterresonanz und Stimmenschalter Bit 0 : 1= Stimme 1 wird über den Filter geleitet Bit 1 : 1= Stimme 2 wird über den Filter geleitet Bit 2 : 1= Stimme 3 wird über den Filter geleitet Bit 4-7 : Filterresonanz (0-15)	54295
24	Lautstärke und Filter an/aus Bit 0-3 : Gesamte Lautstärke Bit 4 : 1= Tiefpaß eingeschaltet Bit 5 : 1= Bandpaß eingeschaltet Bit 6 : 1= Hochpaß eingeschaltet Bit 7 : 1= Stimme 3 wird stumm geschaltet	54296
Die nächsten Register können nur gelesen werden:		
25-26	A/D-Wandler 1 und 2	54297-54298
27	Rauschgenerator-Wert von Stimme 3 augenblicklicher Stand des Rauschgenerators; erzeugt gute Zufallszahlen	54299
28	Hüllkurvengenerator-Wert von Stimme 3 augenblicklicher Stand der Lautstärke von Stimme 3 durch die Hüllkurve; geeignet zur Modulation anderer Parameter	54300

Bild 7. Die Registerbelegung des SID 6581

Register	Bedeutung	Adresse
8	Zehntelsekunden der Echtzeituhr Bits 0-3 Zehntelsekunden	56328
9	Sekunden der Echtzeituhr Bit 0-3 Einersekunden Bit 4-6 Zehnersekunden	56329
10	Minuten der Echtzeituhr Bit 0-3 Einerminuten Bit 4-6 Zehnerminuten	56330
11	Stunden der Echtzeituhr Bit 0-3 Einerstunden Bit 4-6 Zehnerstunden Bit 7 0 = vormittags (AM) 1 = nachmittags(PM)	56331

Bild 8. Notwendige Register des CIA zur Nutzung der Echtzeituhr

allerdings nicht sehr ratsam, da sich sonst der Grafikspeicher mit der wichtigen Zeropage überschneiden würde.

Das Verschieben der eben genannten Bereiche erfordert aber einiges an Programmiererfahrung und sollte nur mit äußerster Vorsicht geschehen.

Damit wäre der Abschnitt über den VIC 6569 abgeschlossen. Sollte Ihnen aufgrund der Informationsfülle der Kopf noch nicht rauchen, wird er es spätestens am Ende des nächsten Abschnittes tun. Denn da nehmen wir den SID 6581 mit seinen Eigenschaften zur Klangerzeugung unter die Lupe.

Der SID 6581 ist ein sehr außergewöhnlicher Baustein. Anders als bei den üblichen Tongeneratoren in Computern, kann er als vollständiger dreistimmiger Mini-Synthesizer bezeichnet werden, da er alle Elemente für die einfache Tonerzeugung bis zur Herstellung komplexer Klänge besitzt.

So können mit ihm drei unabhängige Stimmen frei programmiert werden. Dafür bietet er vier Schwingungsarten (Dreieck, Sägezahn, Rechteck und rosa Rauschen) an, die auch gemischt werden können. Jede Stimme besitzt außerdem einen Hüllkurvengenerator (ADSR-Control), der den zeitlichen Ablauf der Lautstärke eines Tones bestimmt. Alle drei Stimmen lassen sich über drei ebenfalls mischbare Filter leiten, womit sich die Klangfarbe manipulieren läßt. Auch die Möglichkeit der Ringmodulation wurde nicht vergessen.

Der SID ist also ein sehr komplexer Baustein, der sich auch nicht gerade einfach programmieren läßt. Will man komplizierte Klänge erzeugen, bedarf es schon einiger Erfahrung und Kenntnisse im Bereich der Tonsynthese. Wie eine solche Klangerzeugung prinzipiell vor sich geht, soll hier erklärt werden.

Richten wir zunächst wieder einen Blick auf die Register des SID, von denen es deren 28 gibt. Bild 7 zeigt eine Aufstellung aller Register mit den Adressen, über die sie angesprochen werden können.

Alle Funktionen, wie die Tonhöhe, Hüllkurve und Schwingungsformen, sind für jede der drei Stimmen extra vorhanden.

Register 0– 6 Stimme 1
Register 7–13 Stimme 2
Register 14–20 Stimme 3

Die entsprechenden Register sind für jede Stimme identisch, kleine Änderungen sind in der Tabelle in Bild 5 besonders erwähnt.

Untersuchen wir nun die Register für Stimme 1. Dasselbe gilt dann analog für die anderen beiden Stimmen.

Das wichtigste Register für Stimme 1 ist das Register Nummer 4, das sogenannte Steuerregister, mit dem die wichtigsten Funktionen ein- und ausgeschaltet werden können. Durch Setzen und Löschen der entsprechenden Bits auf die schon bekannte Weise lassen sie sich einfach bedienen. Soll für Stimme 1 beispielsweise die Schwingungsform Rechteck eingestellt werden, ist lediglich Bit 6 in Register 4 zu setzen:

```
POKE 54276, PEEK(54276) OR 2^6
```

Das Puls-Pause-Verhältnis ist in Register 2 und 3 einstellbar. Register 5 und 6 geben dem SID die ADSR-Werte für die Hüllkurve in jeweils 4-Bit-Zahlen an. Jeder dieser Parameter kann Werte von 0-15 annehmen und ist einfach mit POKE zu beschreiben. Eine Einstellung des Decay-Wertes auf 7 ist zum Beispiel durch folgenden Befehl möglich:

```
POKE 54277,16*7
```

Damit werden die Bits 4-7 auf den dezimalen Wert 7 gestellt. Ebenso kann man auch Attack, Sustain und Release für jede Stimme programmieren.

Register 0 und 1 bestimmen schließlich die Tonhöhe. Man muß dabei die entsprechende Frequenz des Tones angeben. Welcher Ton welcher Frequenz entspricht, ist in einer Tabelle im C 64-Handbuch aufgelistet.

Wenn gewünscht, können alle drei Stimmen über einen Filter geleitet werden. In Register 23 dienen die Bits 0-2 als Schalter, welche der drei Stimmen gefiltert werden sollen. Zur Einstellung des Filters werden die Register 21-24 verwendet, über die Resonanz- und Filterfrequenz bestimmt werden können.

Mit den Bits 0-3 des Registers 24 läßt sich zusätzlich die Gesamtlautstärke (0-15) aller Stimmen verändern.

POKE 54296,1 sehr leise
POKE 54296,15 sehr laut

Die Register 27 und 28 bilden unter allen anderen Registern, die nur beschrieben werden können, eine Ausnahme. Sie lassen sich nur lesen. Register 27 mit dem Namen Rauschgenerator zeigt bei eingeschalteter Stimme 3 und der gewählten Schwingungsform Rauschen den augenblicklichen Stand des Rauschgenerators an. Da es sich um rosa Rauschen handelt, ist dieser Wert absolut zufällig und kann in Spielen als hervorragender Zufallszahlengenerator verwendet werden.

Durch Bit 7 im Register 24 läßt sich die Stimme 3 stummschalten, das heißt sie ist eingeschaltet, aber nicht zu hören.

```
POKE 54296, PEEK(54296) OR 128
```

schaltet Stimme 3 stumm.

So können die Werte auch ohne hörbare Stimme 3 gelesen und benutzt werden.

Das gleiche gilt auch für Register 28, das den augenblicklichen Stand der Hüllkurvenamplitude von Stimme 3 enthält. Dieser Parameter kann zum Beispiel auch bei stummgeschalteter Stimme 3 hüllkurvenabhängigen Änderungen der Parameter anderer Stimmen dienen, was schier unerschöpfliche Möglichkeiten der Klangerzeugung bietet. Was damit möglich ist, kann man am besten durch intensives Ausprobieren und Setzen und Löschen der einzelnen Bits erfahren. Ein Kochrezept gibt es dafür nicht. Listing 9 erzeugt einen Ton in allen Schwingungsformen. Es soll zeigen, wie die Erzeugung eines Tones prinzipiell abläuft.

Zum Abschluß noch einiges über unsere Ein-/Ausgabe-Bausteine CIA 1 und 2. Sie sind für den Basic-Programmierer nur von geringer Bedeutung und müssen deshalb nur kurz erwähnt werden. Ein Bonbon jedoch gibt es, das sich von Basic aus erreichen läßt: Die eingebaute Echtzeituhr.

Wer schon öfters mit der softwaregestützten C 64-Uhr über die Variable TI$ gearbeitet hat, wird die enorme Ungenauigkeit dieses Zeitgebers bemängelt haben. Für diejenigen, die jedoch genaueste Uhrzeiten benötigen, bietet sich die CIA-Echtzeituhr mit ihrer recht guten Genauigkeit an. Als Taktgeber wird dafür die Netzfrequenz von 50 Hz (in Europa) verwendet, die, wie wir wissen, sehr gut stabilisiert und somit geradezu ideal dafür geeignet ist.

Die Uhr setzt sich aus Zehntelsekunden, Sekunden, Minuten und Stunden zusammen und ist in den Registern 8 bis 11 eines jeden CIA gespeichert. Die für die Uhr wichtige Belegung dieser Register ist in Bild 8 zusammengestellt.

Beim Stellen der Uhr muß jedoch eine bestimmte Reihenfolge eingehalten werden. Schreibt man in das Stunden-Register, bleibt die Uhr stehen und läuft erst wieder an, wenn das Zehntelsekunden-Register beschrieben worden ist. Man sollte also mit dem Stellen bei den Stunden beginnen und die Zehntelsekunden zum Schluß setzen, damit die Uhr auch zum richtigen Zeitpunkt wieder anläuft. In Listing 10 und 11 sind kleine Basic-Programme abgedruckt, die das Stellen und Ausgeben der Echtzeituhr vornehmen. Sie können nach Belieben abgeändert werden, um sie in die eigenen Basic-Programme einzubauen.

Welchen CIA Sie für Ihre Uhr verwenden, ist gleichgültig. Wenn Sie wollen, können Sie beide programmieren und haben somit zwei recht exakte Uhren zur Verfügung. In den beiden Basic-Programmen muß dazu lediglich die Startadresse des gewünschten CIAs eingesetzt werden.

Wie man sieht, lassen sich auch von Basic aus einige Dinge mit den Peripheriebausteinen VIC, SID und den beiden CIAs anstellen. Wichtig dabei ist das direkte Ausprobieren (insbesondere beim SID), um die Wirkung sofort zu erfahren. Dadurch bekommt man Erfahrung und lernt immer weitere Tricks und Kniffe bei der Programmierung von Grafik und Sound.

(Michael Thomas/ks)

```
10 REM FARBDEMO (TEXTMODUS)                    <020>
20 PRINT"{CLR}"                                 <008>
30 FOR X=1024 TO 2023:REM VIDEORAM             <087>
40 POKE 54272+X, INT(RND(1)*16):REM FARBRA
   M VON 54272-55296                           <036>
50 POKE X,INT(RND(1)*128):REM BELIEBIGE BI
   LDSCHIRMWERTE                               <159>
60 NEXT X                                      <008>
70 GET T$:IF T$=""THEN 70                      <145>
```

Listing 1. Farbdemo auf dem Textbildschirm

```
50000 REM GRAFIKPUNKT SETZEN (HI-RES)          <171>
50010 GA=8192:REM STARTADRESSE GRAFIKSPEIC
      HER                                      <228>
50020 AD=320 * INT(Y/8) + (Y AND 7) + (8 *
       INT(X/8)):REM ADRESSE BERECHNEN         <238>
50030 BN= 7-(X AND 7):REM BITNUMMER            <226>
50040 POKE GA+AD, PEEK(GA+AD) OR 2↑BN          <140>
50050 END                                      <011>
```

Listing 2. Setzen eines Grafikpunktes im HiRes-Modus

```
51000 REM GRAFIKPUNKT LOESCHEN (HI-RES)        <096>
51010 GA=8192:REM STARTADRESSE GRAFIKSPEIC
      HER                                      <212>
51020 AD=320 * INT(Y/8) + (Y AND 7) + (8 *
       INT(X/8)):REM ADRESSE BERECHNEN         <222>
51030 BN= 7-(X AND 7):REM BITNUMMER            <210>
51040 POKE GA+AD, PEEK(GA+AD) AND (255-2↑B
      N)                                       <049>
51050 END                                      <251>
```

Listing 3. Löschen eines HiRes-Grafikpunktes

```
51000 REM GRAFIKPUNKT SETZEN (MULTI-COLOR) <178>
51010 GA=8192:REM STARTADRESSE GRAFIKSPEIC
      HER                                      <212>
51020 AD=320 * INT(Y/8) + (Y AND 7) + (8 *
       INT(X/4)):REM ADRESSE BERECHNEN         <093>
51030 B1=(3-(X-(4*INT(X/4))))*2:REM BITNUM
      MER 1                                    <125>
51040 B2=B1+1:REM BITNUMMER 2                  <158>
51050 IF FA =0 THEN F1=0:F2=0                  <021>
51060 IF FA =1 THEN F1=0:F2=1                  <096>
51070 IF FA =2 THEN F1=1:F2=0                  <177>
51080 IF FA =3 THEN F1=1:F2=1                  <252>
51090 IF F1=0 THEN GOTO 51110                  <034>
51100 POKE GA+AD, PEEK(GA+AD) OR 2↑B1          <021>
51110 IF F2=0 THEN GOTO 51130                  <102>
51120 POKE GA+AD, PEEK(GA+AD) OR 2↑B2          <073>
51130 RETURN                                   <131>
51140 REM FA= FARBWERT DES PUNKTES (0-3)       <199>
```

Listing 4. Setzen eines beliebigen Punktes im Multicolor-Bildschirm

```
100 REM SINUSKURVE IN HI-RES                   <006>
110 REM                                        <172>
120 REM GRAFIK EIN                             <222>
130 VIC=53248                                  <015>
140 POKE VIC+17, PEEK(VIC+17) OR 11*16:REM
     GRAFIK EIN                                <060>
150 POKE VIC+22, PEEK(VIC+22) AND 255-16:R
    EM MUTICOLOR EIN                           <154>
155 POKE VIC+24, PEEK(VIC+24) OR 8:REM VIC
    -ARBEITSBEREICH AUF $0000-$3FFF            <153>
160 REM GRAFIKBILD LOESCHEN                    <058>
170 GB=8192:REM STARADRESSE GRAFIK             <204>
180 FOR X=GB TO GB+8000:POKE X,0:NEXT X        <090>
190 REM FARBWERTE SETZEN                       <043>
200 VR=1024:REM STARTADRESSE HIRES-FARBRAM
     (=VIDEORAM)                               <046>
220 FARBE=0*16+14:REM HINTERGRUND HELLBLAU
     PUNKTE SCHWARZ                            <154>
240 FOR X=VR TO VR+1000:POKE X,FARBE:NEXT
    X                                          <075>
251 POKE 53280,0                               <045>
260 REM SINUSKURVE ZEICHNEN                    <215>
270 FOR X=0 TO 319                             <110>
280 Y=70* SIN(X/25.6) +99                      <239>
290 GOSUB 51000                                <064>
300 NEXT X                                     <250>
9999 END                                       <095>
51000 REM GRAFIKPUNKT SETZEN (HIRES)           <075>
51010 GA=8192:REM STARTADRESSE GRAFIKSPEIC
      HER                                      <212>
51020 AD=320 * INT(Y/8) + (Y AND 7) + (8 *
       INT(X/8)):REM ADRESSE BERECHNEN         <222>
51030 BN=7-(X AND 7):REM BITNUMMER BERECHN
      EN                                       <208>
51040 POKE GA+AD, PEEK(GA+AD) OR 2↑BN          <124>
51050 RETURN                                   <051>
```

Listing 5. Dieses Programm zeichnet eine einfarbige Sinuskurve in HiRes auf den Bildschirm


```
100 REM SINUSKURVE IN MULTICOLOR               <238>
110 REM                                        <172>
120 REM GRAFIK EIN                             <222>
130 VIC=53248                                  <015>
140 POKE VIC+17, PEEK(VIC+17) OR 11*16:REM
     GRAFIK EIN                                <060>
150 POKE VIC+22, PEEK(VIC+22) OR 16:REM MU
    TICOLOR EIN                                <183>
155 POKE VIC+24, PEEK(VIC+24) OR 8             <059>
160 REM GRAFIKBILD LOESCHEN                    <058>
170 GB=8192:REM STARADRESSE GRAFIK             <204>
180 FOR X=GB TO GB+8000:POKE X,0:NEXT X        <090>
190 REM FARBWERTE SETZEN                       <043>
200 VR=1024:REM STARTADRESSE VIDEORAM          <043>
210 FR=54272:REM STARADRESSE FARBRAM           <055>
220 F1=2*16+6:REM FARBE 1 ROT UND 2 BLAU       <012>
230 F2=13:REM FARBE 3                          <020>
240 FOR X=VR TO VR+1000:POKE X,F1:NEXT X       <065>
250 FOR X=FR TO FR+1000:POKE X,F2:NEXT X       <073>
251 POKE 53280,0                               <045>
260 REM SINUSKURVE ZEICHNEN                    <215>
270 FOR X=0 TO 159                             <113>
280 Y=70* SIN(X/25.6) +99:FA=INT(RND(1)*4)
    +1                                         <110>
290 GOSUB 51000                                <064>
300 NEXT X                                     <250>
9999 END                                       <095>
51000 REM GRAFIKPUNKT SETZEN (MULTI-COLOR) <178>
51010 GA=8192:REM STARTADRESSE GRAFIKSPEIC
      HER                                      <212>
51020 AD=320 * INT(Y/8) + (Y AND 7) + (8 *
       INT(X/4)):REM ADRESSE BERECHNEN         <093>
51030 B1=(3-(X-(4*INT(X/4))))*2:REM BITNUM
      MER 1                                    <125>
51040 B2=B1+1:REM BITNUMMER 2                  <158>
51050 IF FA =0 THEN F1=0:F2=0                  <021>
51060 IF FA =1 THEN F1=0:F2=1                  <096>
51070 IF FA =2 THEN F1=1:F2=0                  <177>
51080 IF FA =3 THEN F1=1:F2=1                  <252>
51090 IF F1=0 THEN GOTO 51110                  <034>
51100 POKE GA+AD, PEEK(GA+AD) OR 2↑B1          <021>
51110 IF F2=0 THEN GOTO 51130                  <102>
```

```
51120 POKE GA+AD, PEEK(GA+AD) OR 2↑B2          <073>
51130 RETURN                                    <131>
51140 REM FA= FARBWERT DES PUNKTES (0-3)        <199>
```

Listing 6. Nach dem Löschen des Grafik-Bildschirms erscheint eine bunte Sinuskurve in Multicolor-Grafik

```
10 REM SPRITEDEFINITION                         <186>
20 DATA 0,255,0                                 <030>
30 DATA 1,227,192                               <231>
40 DATA 3,255,224                               <183>
50 DATA 7,255,240                               <201>
60 DATA 15,255,248                              <074>
70 DATA 31,255,252                              <204>
80 DATA 63,255,254                              <046>
90 DATA 124,111,254                             <234>
100 DATA 251,223,255                            <114>
110 DATA 251,158,107                            <240>
120 DATA 248,133,167                            <031>
130 DATA 251,108,111                            <079>
140 DATA 251,109,239                            <132>
150 DATA 252,238,111                            <007>
160 DATA 127,255,254                            <029>
170 DATA 63,255,252                             <104>
180 DATA 31,255,248                             <146>
190 DATA 15,255,240                             <076>
200 DATA 7,255,224                              <119>
210 DATA 3,255,192                              <107>
220 DATA 1,255,0                                <238>
225 RESTORE                                     <019>
230 AD=11*64:REM BLOCKADRESSE FUER BLOCK 1
    1                                           <240>
240 FOR Q=AD TO AD+62                           <125>
250 READ SW: POKE Q,SW                          <042>
260 NEXT Q                                      <154>
```

Listing 7. Dieses Programm erzeugt Spritedaten in Block 11 des VIC-Adreßbereichs

```
10 REM SPRITEDEMONSTRATION                      <224>
20 DATA 0,255,0                                 <030>
30 DATA 1,227,192                               <231>
40 DATA 3,255,224                               <183>
50 DATA 7,255,240                               <201>
60 DATA 15,255,248                              <074>
70 DATA 31,255,252                              <204>
80 DATA 63,255,254                              <046>
90 DATA 124,111,254                             <234>
100 DATA 251,223,255                            <114>
110 DATA 251,158,107                            <240>
120 DATA 248,133,167                            <031>
130 DATA 251,108,111                            <079>
140 DATA 251,109,239                            <132>
150 DATA 252,238,111                            <007>
160 DATA 127,255,254                            <029>
170 DATA 63,255,252                             <104>
180 DATA 31,255,248                             <146>
190 DATA 15,255,240                             <076>
200 DATA 7,255,224                              <119>
210 DATA 3,255,192                              <107>
220 DATA 1,255,0                                <238>
225 RESTORE                                     <019>
230 AD=11*64:REM BLOCKADRESSE FUER BLOCK 1
    1                                           <240>
240 FOR Q=AD TO AD+62                           <125>
250 READ SW: POKE Q,SW                          <042>
260 NEXT Q                                      <154>
270 REM SPRITE ANSCHALTEN                       <023>
280 VIC=53248                                   <167>
290 POKE 2040,11:REM SPRITEZEIGER AUF BLOC
    K 11                                        <094>
300 POKE VIC+39,2:REM SPRITEFARBE 0 AUF RO
    T                                           <106>
310 POKE VIC+21,1:REM SPRITE 0 EIN              <091>
320 REM SPRITEBEWEGUNG                          <124>
330 FOR X=20 TO 255                             <083>
340 Y= 199-(ABS(150*SIN(X/20)))                 <243>
350 POKE VIC,X:POKE VIC+1,Y                     <243>
360 NEXT X:GOTO 330                             <159>
```

Listing 8. Hier wird ein definiertes Sprite über den Bildschirm bewegt, wobei der Eindruck eines hüpfenden Balls entsteht

```
10 REM TONDEMO                                  <069>
20 FQ=4440:REM FREQUENZWERT                     <037>
25 FH=INT(FQ/256):FL=FQ-256*FH                  <237>
30 S=54272:REM BASISADRESSE FUER STIMME 1       <142>
35 POKE S+5,0:POKE S+6,15*16                    <077>
36 POKE S+24,15                                 <202>
40 PRINT"DREIECKSCHWINGUNG"                     <197>
50 POKE S,FL:POKE S+1,FH                        <240>
60 POKE S+4,16:REM DREIECK                      <171>
70 POKE S+4,17:REM TON AN                       <208>
80 GET T$:IF T$=""THEN 80                       <188>
90 PRINT"SAEGEZAHNSCHWINGUNG"                   <000>
100 POKE S+4,32:REM SAEGEZAHN                   <076>
110 POKE S+4,33:REM TON AN                      <245>
120 GET T$:IF T$=""THEN 120                     <155>
130 PRINT"RECHTECKSCHWINGUNG"                   <089>
140 POKE S+2,100:POKE S+3,0                     <231>
150 POKE S+4,64:REM RECHTECK                    <076>
160 POKE S+4,65:REM TON AN                      <171>
165 GET T$:IF T$=""THEN 165                     <075>
170 PRINT"RAUSCHEN"                             <232>
180 POKE S+4,128:REM RAUSCHEN                   <042>
190 POKE S+4,129:REM TON AN                     <064>
200 GET T$:IF T$=""THEN 200                     <139>
210 POKE S+4,0:REM TON AUS                      <075>
```

Listing 9. Erzeugung eines Tons in allen vier Wellenformen des SID 6581

```
100 REM ECHTZEITUHR STELLEN                     <005>
110 CIA=56328:REM ANFANGSADRESSE VON CIA 1      <175>
120 PRINT"GEBEN SIE DIE GENAUE UHRZEIT EIN
    "                                           <118>
130 PRINT"{3SPACE}HHMMSS{8LEFT}";:INPUT Z$      <207>
140 HH=VAL(LEFT$(Z$,2))                         <159>
150 MM=VAL(MID$(Z$,3,2))                        <085>
160 SS=VAL(RIGHT$(Z$,2))                        <183>
170 IF HH>23 OR MM>59 OR SS>59 THEN PRINT"
    EINGABEFEHLER":GOTO 120                     <005>
180 IF HH>12 THEN HH=HH-12:TM=1                 <241>
190 POKE CIA+7, PEEK(CIA+7) AND 255-2↑7         <126>
200 POKE CIA+6, PEEK(CIA+6) OR 2↑7              <231>
210 Q= INT(HH/10)*16+HH-INT(HH/10)*10           <133>
220 IF TM=1 THEN Q=Q+128                        <172>
230 POKE CIA+3,Q                                <043>
240 Q= INT(MM/10)*16+MM-INT(MM/10)*10           <086>
250 POKE CIA+2,Q                                <191>
260 Q= INT(SS/10)*16+SS-INT(SS/10)*10           <068>
270 POKE CIA+1,Q                                <084>
280 POKE CIA+0,0: REM UHR STARTEN               <121>
```

Listing 10. Mit diesem Programm können Sie die Echtzeituhr des CIA 1 auf die Sekunde genau stellen

```
300 REM ECHTZEITUHR LESEN                       <128>
310 CIA=56328:REM ANFANGSADRESSE VON CIA 1      <121>
320 HH=PEEK(CIA+3):ZS=PEEK(CIA)                 <240>
330 MM=PEEK(CIA+2)                              <177>
340 SS=PEEK(CIA+1)                              <251>
350 TM=0: IF HH>128 THEN TM=1:HH=HH-128:RE
    M NACHMITTAGS                               <129>
360 HH=INT(HH/16)*10+(HH AND 15)                <221>
370 IF TM=1 AND HH<>12 THEN HH=HH+12            <150>
375 IF TM=0 AND HH=12 THEN HH=0                 <224>
380 MM=INT(MM/16)*10+(MM AND 15)                <240>
390 SS=INT(SS/16)*10+(SS AND 15)                <147>
400 PRINT"{CLR}"HH":"MM":"SS                    <244>
410 GOTO 320                                    <132>
```

Listing 11. Dieses Programm zeigt laufend die aktuelle Zeit der Echtzeituhr an

»Maschinen-Power« mit Basic

Träumen Sie als Basic-Programmierer nicht auch manchmal von so atemberaubenden Geschwindigkeiten, wie sie die Maschinensprache ermöglicht? Wenn ja, dann kommen Sie im folgenden Artikel voll auf Ihre Kosten. Wir werden dem Basic-Interpreter durch die Anwendung von Betriebssystem-Routinen nämlich ganz schön einheizen.

Der C64 ist in der Grundausstattung nicht gerade mit einem herausragenden Basic gesegnet. Will man mehr aus seinem Commodore herausholen, so kann man sich entweder eine der inzwischen zahlreichen Basic-Erweiterungen zulegen, oder (abgesehen davon, überhaupt eine neue Sprache zu erlernen) sich in die Tiefen der Maschinensprache wagen, um sich seine eigenen Befehle und Unterprogramme zu erstellen.

Es gibt jedoch noch eine dritte Möglichkeit, sozusagen den goldenen Mittelweg, nämlich das Ausnutzen der vorhandenen Interpreter- und Betriebssystem-Routinen von Basic aus. Wenn solche Routinen von Maschinensprache-Programmierern erst einmal analysiert worden sind, dann wird auch der Basic-Programmierer in die Lage versetzt, diese Unterprogramme ohne Maschinensprache-Kenntnisse zu verwenden.

Kleine »Byte«ologie

Folgende kleine Tatsachen sollten Ihnen jedoch schon geläufig sein. Wie Sie sicher wissen, speichert Ihr Computer jede Zahl, jeden Buchstaben, jedes Programm, schlicht und einfach alles in Bytes ab. Ein Byte kann bis zu 256 verschiedene Werte annehmen, man kann also Zahlen von 0 bis 255 darin speichern. Um größere Werte verarbeiten zu können, nimmt man einfach zwei Byte zusammen und erhält so die maximale Zahl 65535. Das soll für unsere Zwecke genügen. Das erste Byte enthält den niederwertigen Anteil (nicht, wie man vielleicht annehmen möchte, den höherwertigen Teil), im folgenden deshalb Low-Byte genannt. Entsprechend heißt das zweite, höherwertige Byte, High-Byte.

Folgende Basic-Zeile wandelt eine Zahl in zwei Byte um:

```
H%=INT(X%/256):L%=X%-H%*256
```

```
1 REM BLOCKVERSCHIEBUNG                       <198>
2 REM CREATED BY CHRISTOPH BERGMANN           <242>
3 REM                                         <065>
10 DEF FN H(X)=INT(X/256)                     <244>
20 DEF FN L(X)=X-FN H(X)*256                  <019>
100 INPUT"ALTE STARTADRESSE";S                <215>
110 INPUT"ALTE ENDADRESSE+1";E                <089>
120 INPUT"NEUE ENDADRESSE+1";N                <212>
200 POKE 95,FN L(S):POKE 96,FN H(S)           <221>
210 POKE 90,FN L(E):POKE 91,FN H(E)           <202>
220 POKE 88,FN L(N):POKE 89,FN H(N)           <136>
300 SYS 41920:END                             <098>
```

Listing 1. Routine zur Verschiebung von beliebigen Speicherbereichen

X% ist hierbei die zu wandelnde Zahl; L% und H% natürlich das Low- und High-Byte.

Und umgekehrt das Errechnen einer Zahl aus zwei Byte:

```
X%=H%*256%+L%
```

Eine Eigenschaft der Maschinensprache ist es, bei Angaben von Speicherbereichen die Endadresse plus 1 anzugeben. Beispiel: Sie wollen den Bildschirm von 1024 bis 2047 speichern. Dann müssen Sie als Startdresse den Wert 1024 und als Endadresse den Wert 2048 (!) verwenden.

So, soviel zum nötigen »Grundwissen«. Jetzt geht's los mit dem ersten Tip! Eine häufige Aufgabe, die der Computer zu erfüllen hat, ist das Verschieben von Speicherbereichen. Wollen Sie zum Beispiel einen deutschen Zeichensatz auf dem C64 realisieren, so müssen Sie zuerst den normalen Zeichensatz auf einen freien Platz kopieren und dann dort die gewünschten Zeichen in Umlaute ändern. Eine andere Anwendung ist das Kopieren des Basic-ROMs und des Betriebssystem-ROMs (Kernel) in das darunterliegende RAM, um dort dann Änderungen vorzunehmen (zum Beispiel Eindeutschen des Befehlssatzes). Dabei müssen allerdings über 16000 Byte gelesen und wieder geschrieben werden.

```
1 REM DEMO ZUR CURSORPOSITIONIERUNG            <250>
2 REM CREATED BY CHRISTOPH BERGMANN            <242>
3 REM                                          <065>
10 POKE 53280,0:POKE 53281,0                   <138>
20 PRINT"{CLR}";                               <214>
30 FOR X=0 TO 6 STEP.1                         <036>
40 Y=SIN(X)+1                                  <216>
50 POKE 211,X*6:POKE 214,Y*11                  <184>
60 SYS 58732:POKE 783,1:SYS 58634              <038>
70 PRINT"{WHITE}6{GREY 3}4{GREY 2}E{GREY 1
   }R"                                         <007>
80 NEXT:END                                    <051>
```

Listing 2. Dieses Programm demonstriert die Cursor-Positionierung auf dem Bildschirm

In Basic dauert das eine ganze Weile. Aber der Interpreter stellt uns eine Routine zur Verfügung, die genau dasselbe in Sekundenschnelle für uns erledigt. Hierzu muß man in die Speicherstelle 95 das Low-Byte und in 96 das High-Byte der Anfangsadresse des zu verschiebenden Bereichs angeben, sowie entsprechend in 90 und 91 die Endadresse. In 88 und 89 schließlich muß die Endadresse (!) des Zielbereichs angegeben werden. Rufen Sie danach die Roumit SYS 41920 auf (Das Programm in Listing 1 verschiebt einen beliebigen Speicherblock). In den Zeilen 10 und 20 sehen Sie übrigens eine sehr schöne und komfortable Methode zur Umwandlung einer Adresse in das Low-Byte und High-Byte.

Speicher »herumschieben«

Vom Verschieben von Speicherbereichen zum Verschieben des Bildschirms. Sie können den Bildschirm jederzeit, ohne die Cursorposition zu verändern, nach oben scrollen, indem Sie einfach die entsprechende Routine mit SYS 59626 aufrufen.

An dieser und an den folgenden Routine sehen Sie übrigens, wie leistungsfähig das Betriebssystem des Commodore 64 zum Beispiel in der Bildschirmsteuerung ist. Man muß es nur zu nutzen wissen. Wenn Sie eine oder mehrere Zeilen auf dem Bildschirm löschen wollen, so können Sie dies ebenfalls dem Betriebssystem überlassen. Einfach die zu löschende Zeile in die Speicherzelle 781 POKEn und die entsprechende Routine mit SYS 59903 starten. Schon ist die gewünschte Zeile vom Bildschirm verschwunden. Wollen Sie mehrere Zeilen löschen, so können sie das in einer FOR-NEXT-Schleife Das sieht dann folgendermaßen aus:

```
10 FOR T=A TO E : POKE 781,T : SYS 59903 : NEXT T
```

A ist hierbei die Anfangszeile und E entsprechend die Endzeile. Eine weitere, sehr wichtige Unterroutine ist die »Cursor

Setzen/Holen«. Damit kann man den Cursor auf jede beliebige Position des Bildschirms setzen. Einfach die Zeile in 214 und die Spalte in 211 schreiben und... Halt! Machen Sie nicht den Fehler und rufen Sie, wie schon oft in Zeitschriften geschrieben, sofort die Routine auf. Erstens sollten Sie mit POKE 783,1 die Routine auch auf »Setzen« schalten, und zweitens müssen sie vorher nämlich erst die »Cursorposition berechnen«-Routine aktivieren. Dies geschieht mit SYS 58732. Dann können Sie gefahrlos mit SYS 58634 das erste Unterprogramm starten (Das Programm in Listing 2 demonstriert dies mit einer kleinen Sinuskurve).

Für die zweite Funktion der Unterroutine, nämlich »Cursorposition holen«, müssen Sie lediglich POKE 783,0 und SYS 58634 eingeben, danach steht die Zeile des Cursors in der Speicherzelle 781 und die Spalte in 782. Damit läßt sich zum Beispiel ein Menü, aus dem man mit einem Cursor auswählen kann, sehr gut realisieren.

Sicher ist es Ihnen schon einmal passiert, daß Sie alle möglichen Parameter zur Bildschirmausgabe verändert haben (zum Beispiel Rahmen-, Hintergrundfarbe, Lage des Bildschirms, und so weiter...) und Sie nun nicht mehr wissen, wie die Anfangswerte ausgesehen haben. Abgesehen davon ist es sehr mühsam, das alles »per Hand« wieder zurückzustellen. Durch Aufruf der Routine »Bildschirm-Reset« mit SYS 58648 wird alles wieder in den Ausgangszustand (wie nach dem Einschalten, also dunkelblauer Hintergrund, hellblaue Rahmen- und Zeichenfarbe, und so weiter...) zurückgesetzt. Dies sollten Sie übrigens zu Beginn jedes Programms einmal durchführen, da ja der Benutzer vorher alles verstellt haben könnte.

Wenden wir uns nun ab von der Bildschirmsteuerung, hin zu anderen Betriebssystem-Routinen. Zur Fehlersuche zu gebrauchen ist eine Routine, die die aktuelle Zeilennummer, in der sich das Programm gerade befindet, ausgibt. Durch SYS 48578 erfolgt die Ausgabe »in xxxx«, wobei es sich bei »xxxx« um die aktuelle Zeilennummer handelt.

Das laufende Basic-Programm wird danach ganz normal fortgesetzt.

Wenn Sie nicht Fehler verhindern oder aufspüren, sondern produzieren wollen, so hilft Ihnen die folgende Routine weiter. POKEn Sie einen Wert zwischen 1 und 30 in die Speicherzelle 781 und starten Sie das Unterprogramm mit SYS 42039. Schon bricht Ihr Programm mit der der Nummer entsprechenden Fehlerausgabe ab.

```
1 REM SPEICHERBEREICH ABSPEICHERN              <181>
2 REM CREATED BY CHRISTOPH BERGMANN            <242>
3 REM                                          <065>
10 DEF FN H(X)=INT(X/256)                      <244>
20 DEF FN L(X)=X-FN H(X)*256                   <019>
100 INPUT"STARTADRESSE";S                      <085>
110 INPUT"ENDADRESSE+1";E                      <230>
120 INPUT"NAME{8SPACE}";N$                     <146>
150 SYS(57812)N$,8                             <215>
200 POKE 193,FN L(S):POKE 194,FN H(S)          <206>
210 POKE 174,FN L(E):POKE 175,FN H(E)          <222>
300 SYS 62957:END                              <198>
```

Listing 3. Speichern eines beliebigen Speicherbereichs auf Kassette oder Diskette

Alle Kanäle schließen

Nun wieder zu etwas Nützlichem: Wenn Sie in einem Programm mehrere Ein- oder Ausgabekanäle mit OPEN eröffnet haben, so ist es etwas mühsam, alle Kanäle wieder mit CLOSE zu schließen. Das Betriebssystem hat hierfür ein Unterprogramm, das alle möglichen Kanäle auf einmal schließt. Die Startadresse der Routine lautet 62255.

Wie Sie sicher wissen, kann ein Programm nicht nur mit LOAD"Name",8 (Hier wird das Programm immer ab dem Basic-Anfang, der bei 2049 liegt, geschrieben), sondern auch mit LOAD"Name",8,1 geladen werden. Bei letzterem lädt das Betriebssystem ein Programm an die angegebene Stelle im Speicher. Die Startadresse wird beim SAVEn mitgespeichert. Dies kann man sehr nutzbringend anwenden, denn es muß sich ja nicht unbedingt um ein (in diesem Fall Maschinen-)Programm handeln. Man kann vielmehr jeden beliebigen Speicherbereich speichern, zum Beispiel Sprite-Daten, den Bildschirmspeicher, HiRes-Grafiken, Variablenwerte, etc. Es ergeben sich wirklich viele Anwendungsmöglichkeiten.

Um so einen Bereich zu speichern, müssen Sie folgendes eingeben: Als erstes »SYS(57812)"Name",g« zum Eröffnen des Programmfiles. Wenn Sie für »g« den Wert 8 verwenden, so speichern Sie das Programm auf die Diskette, beim Wert 1 auf Kassette. Danach müssen Sie die Startadresse des Speicherbereichs in die Speicherstellen 193 und 194 schreiben, sowie die Endadresse in 174 und 175. Zum Schluß rufen Sie die Routine zum Speichern mit SYS 62957 auf (Alle Basic-Zeiger bleiben dabei unverändert). Das Programm in Listing 3 speichert einen angegebenen Bereich wie oben beschrieben.

```
1 REM EINGABE - UNTERPROGRAMM                  <250>
2 REM CREATED BY CHRISTOPH BERGMANN            <242>
3 REM                                          <065>
60000 SYS 42336:E$="":Z=512                    <059>
60010 P=PEEK(Z):IF P THEN E$=E$+CHR$(P):Z=
      Z+1:GOTO 60010                           <135>
60020 RETURN                                   <131>
```

Listing 4. Unterprogramm zur Eingabe einer Zeichenkette in eine Stringvariable

Im folgenden ein sehr nützliches Beispiel: Wenn Sie in einem Programm Sprites verwenden, so werden Sie die zugehörigen Daten wahrscheinlich in DATA-Zeilen geschrieben haben, diese dann mit einer FOR-NEXT-Schleife wieder auslesen und an einen bestimmten Speicherplatz POKEn. Das dauert bei vielen Sprites nicht nur sehr lange, es verbraucht auch ungefähr vier- bis fünfmal soviel Platz, wie eigentlich nötig wäre. Wenn Sie die Sprite-Daten allerdings auf Diskette speichern und dann direkt in den Speicher laden, so umgehen Sie beide Nachteile. Dies können Sie am besten folgendermaßen bewerkstelligen:

Laden Sie Ihr »altes« Programm und starten Sie es. Nachdem die Sprite-Daten an die richtige Stelle gePOKEt wurden, können Sie es unterbrechen und löschen. Tippen Sie nun das Programm in Listing 3 ab und starten Sie es. Als Start- und Endadresse geben Sie die entsprechenden Werte für Ihre Sprites ein. Nach Eingabe eines Namens werden die Daten als Programmfile gespeichert. Laden Sie nun abermals Ihr »altes« Programm. Daraus können Sie nun die DATA-Zeilen und die FOR-NEXT-Schleife entfernen. Fügen Sie als erste Zeile folgendes ein:

```
1 IF A=0 THEN A=1 : LOAD "Name",8,1
```

Als »Name« verwenden Sie natürlich den Namen, den Sie beim Speichern angegeben haben. So, jetzt ist Ihr neues Programm fertig und Sie können es speichern.

Wenn Sie sich schon einmal über den INPUT-Befehl des Commodore-Basic geärgert haben, weil er verschiedene Zeichen (zum Beispiel Doppelpunkt, Komma, führende Leerzeichen und so weiter) einfach nicht übernimmt, dann ist hier die Abhilfe: Rufen Sie mit SYS 42336 die Eingaberoutine des Betriebssystems auf. Diese Routine schreibt alle Zeichen in einer logischen Bildschirmzeile (maximal 80 Zeichen) in den Basic-Eingabepuffer ab Adresse 512. Daraus kann man nun mit einer einfachen Schleife die Eingabe einlesen. Das Ende wird mit einem CHR$(0) gekennzeichnet (Das Programm in Listing 4 ist ein Beispiel dafür.)

So, das war's mit den Tips. Viel Spaß und Freude damit.

(Christoph Bergman/ks)

Rechnen in Maschinen-sprache

Praktische Tips und Tricks braucht jeder Programmierer. Wir zeigen Ihnen, wie in Maschinensprache gerechnet wird. Mit vielen Beispielen erklären wir Ihnen ausführlich alles Notwendige – Schritt für Schritt.

In diesem Artikel werden die wichtigsten Routinen des Basic-Interpreters zum Umgang mit Zahlen erläutert, zur Division, Multiplikation und so weiter, und die Nutzung dieser Routinen in Assemblerprogrammen. Sollten Sie bereits von der sogenannten »Fließkommadarstellung« gehört haben, die im Zusammenhang mit diesen Routinen ständig benutzt wird und ziemlich komplex ist: Durch diesen Artikel sollen Sie vor allem den praktischen Umgang mit den Arithmetikroutinen lernen. Zahlenformate werden nur behandelt, soweit Sie für das Verständnis unbedingt erforderlich sind. Wenn Sie mehr über die Fließkommadarstellung wissen wollen, empfehle ich Ihnen den Kurs »Assembler ist keine Alchimie« von H.Ponnath (Sonderheft 8/85, Assembler für Einsteiger und Fortgeschrittene).

Wer benötigt die Arithmetikroutinen?

Die Kenntnis dieser Routinen ist vor allem für zwei Typen von Programmierern interessant. Zum einen für jene »Freaks«, die extreme Anforderungen an Kürze und Geschwindigkeit von Programmen stellen und diese daher vollständig in Maschinensprache beziehungsweise Assembler schreiben.

Solange in reinen Maschinenprogrammen die meistverwendeten Integerzahlen nur addiert oder subtrahiert werden müssen, hält sich der Programmieraufwand noch in Grenzen. Die meisten Assemblerprogrammierer werden mir jedoch zustimmen, daß bereits die Multiplikation und Division von Integerzahlen einen recht hohen Aufwand erfordert und es unumgänglich wird, eigene Unterprogramme zur Multiplikation, Division oder zur Ausgabe von Integerzahlen auf dem Bildschirm zu schreiben.

Beim Rechnen mit Integerzahlen können diese Unterprogramme recht sinnvoll sein. Wenn Sie in Ihren Maschinenprogrammen jedoch auch mit Realzahlen arbeiten, ist es unsinnig, auch dafür eigene Routinen zu erstellen. Alle (!) Routinen, die zum Umgang mit Realzahlen benötigt werden, befinden sich bereits in unserem Computer und zwar im Basic-Interpreter, der sie bei der Programmabarbeitung ständig benötigt.

Die zweite Gruppe sind Programmierer, deren Programme vorwiegend in Basic geschrieben werden, die jedoch Unterroutinen in Maschinensprache als Ersatz für besonders zeitkritische Basic-Programmteile einsetzen. Wenn in solchen Routinen gerechnet, zum Beispiel multipliziert werden muß, ist es unsinnig, nach Basic zurückzukehren und die benötigte Rechnung dort vorzunehmen, um anschließend das errechnete Ergebnis in Maschinensprache weiterzuverarbeiten.

Durch die Übergabe der Zahlen an das Basic-Programm (Lesen mit: PEEK(Zahl1), PEEK(Zahl2)) und die Rückgabe des Ergebnisses an die Maschinenroutine mit POKE(Adresse), (Ergebnis) würde derart viel Zeit vergehen, daß der Geschwindigkeitsvorteil der Maschinenroutine in vielen Fällen zunichte gemacht wird. Die Berechnungen müssen in solchen Fällen ebenfalls im Maschinenprogramm durchgeführt werden.

Bei gemischten Basic- und Assemblerprogrammen stellt sich oft das Problem der Parameterübergabe und des Zugriffs auf eine Basic-Variable von einem Maschinenprogramm aus. Ein Beispiel: Eine Maschinenroutine soll die Werte eines Integer- oder Realarrays aufsummieren. Das erste Problem besteht in der Parameterübergabe: Wie wird der Maschinenroutine durch das Basic-Programm der Name des Arrays übermittelt, dessen Werte aufsummiert werden sollen? Das zweite Problem besteht darin, daß die Summierungsroutine »herausfinden« muß, wo im Computerspeicher das Array durch den Basic-Interpreter abgelegt wurde.

Nach der Besprechung der Arithmetikroutinen werde ich erläutern, welche Möglichkeiten es zur Parameterübergabe an ein Maschinenprogramm gibt, und wie der Zugriff auf Basic-Variablen von Maschinensprache aus möglich ist.

Gleich und doch nicht gleich

Kompatibilitätsprobleme zwischen C 16, C 64 und C 128

Wie Sie vielleicht wissen, enthält das Betriebssystem von Commodore-Computern eine sogenannte »Kernel-Sprungtabelle«, eine Liste der wichtigsten Routinen des Betriebssystems und der jeweiligen Einsprungadresse. Diese Sprungtabelle gewährleistet eine problemlose Übertragbarkeit von Assemblerprogrammen, die Betriebssystemroutinen verwenden, zwischen verschiedenen (Commodore-) Computern. Unabhängig davon, ob Sie einen C 16, einen C 64 oder einen C 128 besitzen, können Sie mit der Befehlsfolge:

```
CLC
LDX #5 (Zeile)
LDY #5 (Spalte)
JSR $FFF0 (PLOT, setzt den Cursor)
```

den Cursor immer direkt auf Spalte fünf von Zeile fünf setzen. Die Kompatibilität wird dadurch hergestellt, daß erst nach dem Einsprung in $FFF0 der Sprung zur eigentlichen Routine erfolgt. Zum Beispiel befindet sich beim C 64 an Adresse $FFF0 der Befehl »JMP $E50A«, ein Sprungbefehl mit der C 64-spezifischen Adresse der Routine PLOT.

Diese Sprungtabelle enthält jedoch keinerlei Routinen des Basic-Interpreters, die im folgenden fast ausschließlich ver-

wendet werden. Die folgenden Programme beziehen sich alle auf den C 64. Am Ende dieses Kapitels befindet sich eine Übersicht der verwendeten Routinen und Adressen. Soweit sie mir bekannt sind, finden Sie in dieser Tabelle auch die entsprechenden Adressen beim C 16 und C 128, so daß der Großteil der vorgestellten Listings auch für den C 16 oder den C 128 umgeschrieben werden kann, da die Funktionsweise der Arithmetikroutinen trotz unterschiedlicher Adressen auf allen drei Computern identisch ist. Alle (!) verwendeten Routinen besitzen eine Unterprogrammform, enden also mit einem »RTS« und können problemlos mit »JSR $....« aufgerufen werden.

Die Redaktion ist dankbar für jeden Tip eines Lesers, der Lücken in dieser Tabelle füllen kann, oder aber gar weitere Arithmetikroutinen in seinem Computer entdeckt hat.

Zahlendarstellung im Integer- und Fließkommaformat

In Assemblerprogrammen wird vorwiegend mit Integerzahlen gearbeitet, das heißt mit ganzen Zahlen. Wie groß die darzustellenden Zahlen sein können, hängt davon ab, wie viele Bytes zur Darstellung verwendet werden. Bei Verwendung eines Bytes für eine Integerzahl ergibt sich ein Wertebereich von 0 bis 255, das heißt insgesamt 256 = 2^8 verschiedene ganze Zahlen können dargestellt werden.

Bei Verwendung von zwei Byte = Word = 2^{16} können bereits 65536 verschiedene Zahlen dargestellt werden (0 bis 65535). In fast allen Assemblerprogrammen ist die Verwendung von Zwei-Byte-Zahlen und der sich daraus ergebende Wertebereich völlig ausreichend.

Dieser Wertebereich kann übrigens auch negative Zahlen umfassen, wenn die sogenannte »vorzeichenbehaftete Darstellung« verwendet wird. Bei dieser Art der Darstellung dient das oberste Bit zur Darstellung des Vorzeichens.

```
1000 0000 0000 0000
```

Ist es gelöscht, handelt es sich um eine positive, ist es gesetzt, um eine negative Zahl. Da das als Vorzeichen verwendete Bit zur Angabe der absoluten Größe entfällt, beträgt die größte mit zwei Byte darstellbare vorzeichenbehaftete Zahl 32767. Da der Wertebereich auch auf negative Zahlen ausgeweitet wird, können alle Zahlen zwischen -32768 und +32767 dargestellt werden.

```
..+ 32767 = 7fff = 0111 1111 1111 1111
  + 32766 = 7ffe = 0111 1111 1111 1110
  + 32765 = 7ffd = 0111 1111 1111 1101
    ...     ...        ...
  +     1 = 0001 = 0000 0000 0000 0001
        0 = 0000 = 0000 0000 0000 0000
  -     1 = ffff = 1111 1111 1111 1111
    ...     ...        ...
  - 32766 = 8002 = 1000 0000 0000 0010
  - 32767 = 8001 = 1000 0000 0000 0001
..- 32768 = 8000 = 1000 0000 0000 0000
```

Die zweite Art der Zahlendarstellung, das Fließkommaformat, erlaubt die Darstellung beliebiger Realzahlen, der Wertebereich ist daher extrem groß im Vergleich mit der üblichen Zwei-Byte-Integerdarstellung. Alle (!) Zahlen, auch Integerzahlen, wandelt der Basic-Interpreter übrigens zuerst in Fließkommazahlen, bevor arithmetische Operationen mit ihnen ausgeführt werden.

Dennoch ist es keineswegs notwendig, daß Sie wissen, wie Zahlen im Fließkommaformat dargestellt werden. Zur Umwandlung der in Maschinenprogrammen üblichen Integerzahlen ins Fließkommaformat existieren Routinen, die noch von mir besprochen werden.

Vorläufig genügt es, wenn Sie sich merken, daß der Basic-Interpreter Fließkommazahlen in Form von fünf Byte im Speicher ablegt, wobei ein Byte für den »Exponenten« und vier Byte für die »Mantisse« verwendet werden. Das Vorzeichen der Zahl wird mit dem obersten Bit eines der vier Mantissenbyte dargestellt. Ist das Bit gelöscht, ist die Zahl positiv, bei gesetztem Bit ist sie negativ.

Die Fließkomma-Akkumulatoren

Die arithmetischen Operationen des Basic-Interpreters finden in den beiden sogenannten »Fließkomma-Akkumulatoren« statt, in FAC #1 und FAC #2, die meistens FAC und ARG genannt werden. Es handelt sich dabei um zwei Speicherbereiche in der Zeropage, die zur Aufnahme der beiden Zahlen verwendet werden, die miteinander verknüpft werden sollen.

Aufbau von FAC und ARG (C 64)

	FAC	ARG
Exponent	$61	$69
Mantisse 1	$62	$6A
Mantisse 2	$63	$6B
Mantisse 3	$64	$6C
Mantisse 4	$65	$6D
Vorzeichen	$66	$6E

Adressenvergleich

	C 64	C 16	C 128
FAC	$61-$66	$61-$66	$63-$68
ARG	$69-$6E	$69-$6E	$6A-$6F

Der Aufbau von FAC und ARG ist bei allen drei Computern identisch.

Wie die Tabelle zeigt, sind beim C 16 und dem C 64 sogar die Adressen identisch. Vielleicht wundern Sie sich über die zusätzliche Speicherstelle für das Vorzeichenbyte, da ich vor kurzem noch behauptet habe, das Vorzeichen sei in einem Bit der Mantisse »versteckt«: FAC und ARG haben eine etwas verschwenderischere Art der Fließkommadarstellung.

Der Rechner im Computer

Im gesamten Speicher bis auf FAC und ARG werden Fließkommazahlen tatsächlich im kompakten Fünf-Byte-Format abgelegt. Wenn nun eine Fließkommazahl, zum Beispiel die Basic-Variable V1, zur Vorbereitung einer Rechnung nach FAC oder ARG kopiert wird, wird die komplette Mantisse1 nach $66 beziehungsweise nach $6E in das Vorzeichenbyte kopiert, und das oberste Bit von Mantisse1 immer (!) gesetzt (mit ORA #$80), bevor dieses Byte an seine Position im FAC transportiert wird.

Wenn umgekehrt eine Fließkommazahl in FAC oder ARG an eine beliebige Speicheradresse transportiert werden soll, wird das oberste Bit von Mantisse1 aus dem Vorzeichenbyte rekonstruiert, bevor der Transport erfolgt (FAC: LDA $66:ORA #$7F:AND $62).

Diesen Unterschied zwischen dem kompakten Fünf-Byte-Format, das im Speicher verwendet wird, und dem speziellen FAC/ARG-Format müssen Sie unbedingt berücksichtigen, wenn Sie Fließkommazahlen »per Hand« zwischen FAC oder ARG und dem restlichen Speicher hin- und hertransportieren wollen. Wenn ein solches Transferproblem in Ihren Programmen auftritt, und Sie die dafür vorhandenen Routinen nicht verwenden können, weil die entsprechenden Adressen für Ihren Computer in der Tabelle fehlen, müssen Sie beim Kopieren auf dieses Vorzeichenbit beziehungsweise -byte unbedingt achten.

Wenn zwei Zahlen miteinander verknüpft werden sollen (zum Beispiel 10 * 3 oder 10/3), müssen beide Zahlen zuerst ins Fließkommaformat konvertiert werden. Anschließend müssen die Zahlen nach FAC und ARG übertragen werden. Im letzten Schritt wird die jeweilige Arithmetikroutine aufgerufen, zum Beispiel die Multiplikations- oder die Divisionsroutine. Bei allen Arten der Verknüpfung befindet sich das Ergebnis anschließend im FAC.

Bevor nun verschiedene Assemblerprogramme folgen, will ich kurz die wichtigsten Eigenarten des von mir verwendeten MAE-Assemblers beschreiben:

MAE	Hypra-Ass	Bedeutung
.BA $c000	.BA	Startadresse eines Programms
.OS	.OB	Anweisung, Objektcode zu generieren
.FAC.de $61	.EQ FAC=$61	Zuweisung eines Wertes
.LDA #L,FAC	LDA #<(....)	Low-Byte-Wert eines Wertes
.LDA #H,FAC	LDA #<(....)	High-byte-Wert eines Wertes
.BY	.BY	Byte

Ausgabe von Fließkommazahlen

Für alle folgenden Beispielprogramme benötigen wir eine Möglichkeit, das aus der Verknüpfung von zwei Zahlen resultierende Ergebnis möglichst problemlos überprüfen zu können, möglichst eine Ausgabe des Ergebnisses direkt auf dem Bildschirm.

Der Basic-Interpreter stellt zwei Routinen zur Verfügung, die kombiniert zur Ausgabe einer Zahl (die sich im Fließkommaformat im FAC befinden muß) auf dem Bildschirm führen. Die betreffende Zahl wird nach dem Aufruf der beiden Routinen an der momentanen Cursorposition ausgegeben.

FAC-Inhalt in ASCII-String wandeln ($BDDD)

Bevor diese Routine (FACSTR) aufgerufen wird, muß dafür gesorgt werden, daß sich die auszugebende Zahl im Fließkommaformat im FAC befindet. Beim Aufruf der Routine müssen keine weiteren Parameter übergeben werden, das heißt Akku, X- und Y-Register können beliebige Inhalte besitzen. Die Routine wandelt die Fließkommazahl in einen ASCII-String um, der mit Hilfe einer weiteren Routine auf dem Bildschirm ausgegeben werden kann.

Achtung! Die Fließkommazahlen im FAC werden durch die Umwandlung zerstört. Wenn Sie nach der Ausgabe mit diesen Zahlen weiterrechnen wollen, müssen Sie vor Aufruf von FACSTR den Inhalt des FAC in einen freien Speicherbereich kopieren.

ASCII-String ausgeben ($AB1E)

Auch diese Routine (STROUT) benötigt keine weiteren Übergabeparameter. Der Aufruf mit »JSR $AB1E« genügt zur Ausgabe der zuvor in einen String umgewandelten Zahl.

Zahlenausgabe auf dem Bildschirm

```
Eingabe:
        FAC          = Wert
        Akku         = unbenutzt
        X-Register   = unbenutzt
        Y-Register   = unbenutzt

jsr FACSTR           ; $bddd
jsr STROUT           ; $ab1e

Ausgabe:
        Akku         = unbenutzt
        X-Register   = unbenutzt
        Y-Register   = unbenutzt
        FAC          = Inhalt zerstört
```

Konstante im Fließkommaformat nach FAC übertragen ($BBA2)

Um die Wirkung beider Routinen zu demonstrieren, benötigen wir eine Fließkommazahl im FAC. Beim C 64 befindet sich an Adresse $BBA2 die Routine KONFAC, die eine beliebige Konstante im Fließkommaformat in den FAC überträgt.

Diese Routine konvertiert die Zahl zugleich vom kompakten Fünf-Byte-Speicherformat in das Sechs-Byte-Format der Fließkommaakkus (ein zusätzliches Byte für das Vorzeichen und Setzen des obersten Bits von Mantisse1). Die entsprechenden Adressen beim C 16 und C 128 sind mir leider nicht bekannt. Wenn Sie diese Routine in Ihrem Computer nicht finden sollten, können Sie sie jedoch problemlos durch eine Schleife ersetzen, mit der Sie die fünf Byte »per Hand« in den FAC kopieren (vergessen Sie nicht, Mantisse1 in das Vorzeichenbyte zu kopieren und das oberste Bit dieser Mantisse im FAC zu setzen!).

Vor dem Aufruf dieser Routine müssen Akku und Y-Register mit der Adresse geladen werden, an der sich die betreffende Konstante befindet (Akku=Low-Byte/Y=High-Byte).

FAC mit Konstante laden

```
Eingabe:
        Akku         = Low-Byte-Adresse
        X-Register   = unbenutzt
        Y-Register   = High-Byte-Adresse
lda #<(Wert)         ; Low-Byte-Adresse
ldy #>(Wert)         ; High-Byte-Adresse
jsr KONFAC           ; $BBA2
Ausgabe:
        Akku         = unbenutzt
        X-Register   = unbenutzt
        Y-Register   = unbenutzt
        FAC          = Konstante
```

Bildschirmausgabe einer Fließkommazahl

Im Programm »Zahlenausgabe« (Listing 1) wird die Zahl PI (3,14...) verwendet, deren Fließkommadarstellung lautet: $82 $49 $0F $DA $A1.

Das Programm befindet sich an Adresse $C000, die Zahl PI am Programmende. Nach dem Aufruf des Programms mit »SYS 49152« wird der Akku mit dem Low- und das X-Register mit dem High-Byte der Adresse geladen, an der sich die Zahl befindet. Anschließend wird die Routine KONFAC aufgerufen, die die Konstante in den FAC überträgt.

Im FAC befindet sich nun die Zahl PI in der benötigten Form. Durch den Aufruf von FACSTR wird der FAC-Inhalt in einen ASCII-String gewandelt, der mit der Routine STROUT auf dem Bildschirm an der momentanen Cursorposition ausgegeben wird (3,14159265).

FAC als Konstante speichern

FACKON ($BBD4) ist die Umkehrung von KONFAC und kopiert den Inhalt des FAC an eine übergebene Adresse, wobei gleichzeitig die Konvertierung in das gepackte Format vorgenommen wird. Im Gegensatz zu KONFAC wird die Adresse im X- und Y-Register übergeben (X=Low-Byte/Y=High-Byte).

FAC als Konstante speichern

```
Eingabe:
        FAC          = Konstante
        Akku         = unbenutzt
        X-Register   = Low-Byte-Adresse
        Y-Register   = High-Byte-Adresse
jsr FACKON           ; $bbd4
Ausgabe:
        Akku         = unbenutzt
        X-Register   = unbenutzt
        Y-Register   = unbenutzt
```

Konvertierungsroutinen

In der Praxis werden Sie häufig nicht mit Fließkomma-, sondern mit Integerzahlen arbeiten. Um Arithmetikoperationen mit zwei Integerzahlen durchzuführen, müssen diese jedoch zuvor in das Fließkommaformat gewandelt werden. Der Basic-Interpreter besitzt glücklicherweise sowohl Routinen zur Wandlung von Integerzahlen ins Fließkommaformat als auch zur Wandlung von Fließkommazahlen in das Integerformat.

Die wichtigsten dieser »Konvertierungsroutinen« werde ich im folgenden besprechen und zwar:

1. Ein-Byte-Integer nach Fließkomma wandeln.
2. Zwei-Byte-Integer nach Fließkomma wandeln.
3. Fließkomma nach Integer wandeln.

Ein-Byte-Integer (positiv) nach Fließkomma ($B3A2)

Meist wird bei der Verwendung von Integerzahlen die positive Darstellung verwendet, das heißt, daß der Ein-Byte-Wert als positive ganze Zahl zwischen 0 und 255 aufgefaßt wird. Mit der Routine EINPOS wird eine solche Zahl in das Fließkommaformat umgewandelt. Die entsprechende Fließkommazahl befindet sich nach dem Aufruf im FAC. Vor dem Aufruf der Routine muß das Y-Register mit der zu konvertierenden Zahl geladen werden.

Das Programm »Ein-Byte-Integer« (Listing 2) verwendet diese Routine zur Konvertierung der Zahl $80 (dezimal: 128; binär: 10000000). Das Y-Register wird mit $80 geladen und die Routine aufgerufen. Zur Überprüfung der Konvertierung wird der Inhalt des FAC anschließend mit den Routinen FACSTR und STROUT in einen ASCII-String gewandelt und auf dem Bildschirm ausgegeben.

Ein-Byte-Werte einlesen ohne Vorzeichen

```
Eingabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = Wert

ldy #(Wert)     ;
jsr EINPOS      ; $b3a2

Ausgabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Wert
```

Ein-Byte-Integer (vorzeichenbehaftet) nach Fließkomma ($BC3C)

Wenn Sie das Programm »Ein-Byte-Integer« (Listing 2) mit Ihrem Assembler eingeben und mit »SYS 49152« starten, stellen Sie fest, daß auf dem Bildschirm zwei Zahlen unmittelbar hintereinander ausgegeben werden: 128 und -128. Für die Ausgabe der Zahl -128 ist der zweite Programmteil verantwortlich, der die Routine EINNEG ($BC3C) zur Wandlung des Integerwertes $80 verwendet. Diese Routine konvertiert einen vorzeichenbehafteten Ein-Byte-Integer-Wert in eine Fließkommazahl.

Was unter vorzeichenbehaftet zu verstehen ist, wurde bereits kurz angedeutet: Das oberste Bit der Zahl wird als Vorzeichen verwendet. Ist es gesetzt, handelt es sich um eine negative, ansonsten um eine positive Zahl. Da dieses Bit bei der Zahl $80 gesetzt ist, interpretiert die Routine EINNEG $80 als die negative Zahl -128. Zur Verdeutlichung: Vorzeichenbehaftet entspricht $81 dem Wert -127, $82 dem Wert -126 und so weiter.

Ein-Byte-Werte einlesen mit Vorzeichen

```
Eingabe:
        Akku        = Wert
        X-Register  = unbenutzt
        Y-Register  = unbenutzt

lda #(Wert)     ;
jsr EINNEG      ; $bc3c

Ausgabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Wert
```

Beachten Sie bitte, daß vor Benutzung von EINNEG der Akku mit dem jeweiligen Wert geladen werden muß, im Gegensatz zur Routine EINPOS, der die Zahl im Y-Register übergeben wird.

Zwei-Byte-Integer (positiv) nach Fließkomma ($BC49)

In vielen Assemblerprogrammen sind Ein-Byte-Werte unzureichend zur Darstellung der benötigten Zahlen. Ein Programm, das zum Beispiel mit hochauflösender Grafik arbeitet, muß mit Zwei-Byte-Integerzahlen arbeiten, um beliebige Bildschirmpositionen erfassen zu können, da die Grafikauflösung in der Y-Richtung 320 Punkte beträgt und der Wert 255 – die Obergrenze der Ein-Byte-Darstellung – daher überschritten werden kann.

Mit Hilfe der Routine ZWEPOS können positive Zwei-Byte-Integerzahlen (0...65535) in die entsprechende Fließkommazahl konvertiert werden. Vor dem Aufruf wird die jeweilige Zahl nach $62/$63 übertragen (Achtung: $62=High und $63=Low, also zuerst High- und dann Low-Byte!), das X-Register direkt mit $90 geladen und das Carry-Flag gesetzt. Das Programm »Zwei-Byte-Integer« (Listing 3) zeigt die Anwendung dieser Routine zur Konvertierung und Ausgabe (mit FACSTR und STROUT) der Zahl $FFFF.

Zwei Byte einlesen ohne Vorzeichen

```
Eingabe:
        Akku        = unbenutzt
        X-Register  = $90
        Y-Register  = unbenutzt
        Carry       = gesetzt

lda #<(Wert)    ; Low-Byte-Wert
sta $63         ; FAC 2.Mantisse
lda #>(Wert)    ; High-Byte-Wert
sta $62         ; FAC 1.Mantisse
sec             ; Carry set
jsr ZWEPOS      ; $bc49

Ausgabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Wert
```

Zwei-Byte-Integer (vorzeichenbehaftet) nach Fließkomma ($BC44)

Nach dem Aufruf mit »SYS 49152« gibt dieses Programm ebenso wie das Programm »Ein-Byte-Integer« zwei Zahlen unmittelbar hintereinander aus, und zwar die Zahlen 65535 und -1. Der erste Programmteil benutzt die besprochene Routine ZWEPOS zur Ausgabe von $FFFF ($FFFF=65535), der zweite Teil die Routine ZWENEG ($BC44). ZWENEG faßt die übergebene Zahl als vorzeichenbehafteten Wert auf. In dieser Darstellungsart entspricht $FFFF der größten mit zwei Byte darstellbaren negativen ganzen Zahl (nicht der größten darstellbaren positiven, da das gesetzte oberste Bit eine negative Zahl anzeigt), das heißt der Zahl -1.

Die Übergabeparameter dieser Routine sind mit ZWEPOS identisch, abgesehen davon, daß das Carry-Flag diesmal nicht gesetzt, sondern gelöscht werden muß. Die Zahl wird ebenfalls in $62/$63 (High/Low) übergeben, das X-Register mit $90 geladen und das Carry-Flag gelöscht. Wie Sie vielleicht bemerkt haben, ensprechen die Speicherzellen, in denen der Zwei-Byte-Wert übergeben wird, den ersten beiden Bytes der Mantisse des FAC. Beim C 16 beziehungsweise beim C 128 wird der gleiche Teil des FAC zur Übergabe benutzt. Die entsprechenden Speicherstellen sind daher

ebenfalls $62/$63 beim C 16 und $64/$65 beim C 128. Die entsprechenden Einsprungadressen finden Sie in der Tabelle am Ende dieses Artikels.

Zwei Byte einlesen mit Vorzeichen

```
Eingabe:
        Akku        = unbenutzt
        X-Register  = $90
        Y-Register  = unbenutzt
        Carry       = gelöscht

lda #<(Wert)        ; Low-Byte-Wert
sta $63             ; FAC 2.Mantisse
lda #>(Wert)        ; High-Byte-Wert
sta $62             ; FAC 1.Mantisse
clc                 ; Carry clear
jsr ZWENEG          ; $bc44

Ausgabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Wert
```

Fließkomma nach Integer ($BC9B)

In allen drei Computern befinden sich zusätzlich Routinen zur Umwandlung von Drei- und Vier-Byte-Integerzahlen ins Fließkommaformat. Da diese jedoch nur selten in Assemblerprogrammen verwendet werden, verzichte ich auf eine ausführliche Darstellung dieser Routinen.

Wir benötigen jedoch unbedingt eine Routine, die den umgekehrten Weg geht, das heißt, die eine Fließkommazahl ins Integerformat wandelt, da es nach verschiedenen Rechenoperationen mit Fließkommazahlen oft einfacher ist, mit dem Ergebnis in Integerform weiterzuarbeiten, als die in Assembler schwer zu verarbeitende Fließkommaform zu verwenden.

Die Routine FLIINT wandelt jede beliebige Fließkommazahl ins Integerformat um. Beachten Sie jedoch bitte, daß die Wandlung nur bei Fließkommazahlen kleiner als 231 (= 2.14 * 109) fehlerfrei funktioniert. Der Grund dafür ist die begrenzte Byteanzahl, die zur Aufnahme des Integerergebnisses verwendet wird (vier Byte).

Da mit Integerzahlen nur ganze Zahlen dargestellt werden können, werden Nachkommastellen bei der Wandlung abgeschnitten (aus 5249,57 wird daher 5249).

FAC in Integer wandeln

```
Eingabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt

jsr FLIINT          ; $bc9b

Ausgabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Inhalt zerstört
```

Das Programm »Fließkomma« (Listing 4) wandelt eine Fließkommazahl in einen Integerwert und gibt ihn auf dem Bildschirm aus. Als Fließkommazahl wird PI verwendet und mit der Routine KONFAC in den FAC übertragen. Der Aufruf von FLIINT wandelt diese Fließkommazahl ins Integerformat. FLIINT benötigt außer der zu konvertierenden Fließkommazahl im FAC keine weiteren Übergabeparameter.

Das Integerergebnis befindet sich ebenfalls im FAC, in den vier Byte der Mantisse. In $65 wird das niederwertigste, in $62 das höchstwertige Byte des Ergebnisses abgelegt. Ergibt die Wandlung zum Beispiel die Zahl $20FA, befindet sich in $65 der Wert $FA und in $64 der Wert $20.

Da die Integerwandlung von PI das ganzzahlige Ergebnis drei liefert, interessiert uns nur das niederwertigste Byte $65. Zur Ausgabe des Ergebnisses wird das Y-Register mit diesem Byte geladen, die Routine EINPOS zur Wandlung nach Fließkomma verwendet und diese mit FACSTR und STROUT ausgegeben.

Dieser Weg ist zweifellos sehr umständlich. Wenn Sie sich unmittelbar davon überzeugen wollen, daß die Routine FLIINT die Zahl PI in die Integerzahl drei konvertiert und dabei die Nachkommastellen abschneidet, so fügen Sie im Programm »Fließkomma« nach dem Befehl »JSR FLIINT« bitte ein »BRK« ein. Nach dem Start mit »SYS 49152« verzweigt das Programm nach der Wandlung sofort in den Monitor (den Sie zuvor natürlich laden und aktivieren müssen!). Schauen Sie sich mit dem Monitor anschließend $62 bis $65 an, die Mantisse des FAC. Sie werden feststellen, daß $65 den Wert drei (=$03) enthält und die höherwertigen Ergebnisbytes den Wert 0.

Direkte Bildschirmausgabe von positiven Integerzahlen ($BDCD)

Die Routine INTOUT erlaubt die direkte Ausgabe von positiven Zwei-Byte-Integerzahlen auf dem Bildschirm. Die Integerzahl wird in den Registern übergeben (X=Low-Byte; Akku=High-Byte). Es handelt sich dabei keineswegs um eine neue Routine, da die dieser Adresse folgenden Befehle nur die bekannten Routinen ZWEPOS, FACSTR und STROUT nacheinander aufrufen.

Das Programm »Integerausgabe« (Listing 5) demonstriert den Einsatz dieser Routine. Das X-Register wird mit dem Low-Byte und der Akku mit dem High-Byte von $F02A geladen. Der Aufruf von INTOUT führt unmittelbar zur Ausgabe der Zahl 61482.

Diese Routine kann in den verschiedensten Assemblerprogrammen vielseitig verwendet werden (Textverarbeitung: Ausgabe von Seite, Zeile und Spalte; Spiele: Ausgabe der Punktzahl, des Levels und so weiter).

Ausgabe von Zwei-Byte-Werten

```
Eingabe:
        FAC         = Integerzahl
        Akku        = High-Byte-Wert
        X-Register  = Low-Byte-Wert
        Y-Register  = unbenutzt

lda (Wert)          ; FAC 4.Mantisse $65
ldx (Wert)          ; FAC 3.Mantisse $64
jsr INTOUT          ; $bdcd

Ausgabe:
        Akku        = unbenutzt
        X-Register  = unbenutzt
        Y-Register  = unbenutzt
        FAC         = Inhalt zerstört

Beispiel:
Übergabe der Zahl in das Register
lda #$1
ldx #$20
jsr INTOUT
Ergebnis:
288 auf dem Bildschirm
```

Grundrechenroutinen

Nachdem wir nun alle häufig benötigten Konvertierungs- und Ausgaberoutinen kennen, können wir uns den eigentlichen Arithmetikroutinen zuwenden, und zwar zuerst den vier Grundrechenarten.

Die vier vorgestellten Routinen arbeiten prinzipiell gleich: Zwei Fließkommazahlen werden durch einen der Operatoren »+-*/« miteinander verknüpft. Die beiden Zahlen müssen sich vor dem Aufruf der benötigten Routine im FAC beziehungsweise ARG befinden. Das Ergebnis der Operation befindet sich nach dem Aufruf immer im FAC (im Fließkommaformat).

Sehr wichtig ist folgende Bedingung: Vor dem Aufruf einer der Routinen muß sich im Akku der Exponent des FAC befinden ($61). Wir werden Routinen kennenlernen, die für den Transport von Konstanten in den FAC oder das Kopieren des ARG-Inhalts in den FAC zuständig sind. Nach der Rückkehr aus diesen Routinen befindet sich der FAC-Exponent immer im Akku. Achten Sie bitte darauf, daß dieser Akkuinhalt bis zum Aufruf der gewünschten Arithmetikroutine auch erhalten bleibt und nicht durch LDA..., TXA etc. zerstört wird.

Das Programm »Grundrechnen« (Listing 6) demonstriert die Anwendung der vier Grundrechenarten. In diesem Programm wird die noch nicht erwähnte Routine FACARG ($BC0C) verwendet, die eine im FAC enthaltene Fließkommazahl nach ARG kopiert. Die entsprechende Adresse dieser Routine für den C 16 finden Sie am Ende des Artikels. Sollten Sie einen C 128 besitzen, können Sie diese Routine entweder selbst im ROM suchen, oder aber eine Schleife verwenden und den Inhalt des FAC »per Hand« nach ARG kopieren. Sie verzichten dann jedoch auf eine exakte Rundung der Zahl, die die Routine FACARG durchführt, bevor die gerundete Zahl nach ARG kopiert wird.

FAC ins ARG kopieren

```
Eingabe:
      FAC          = Wert
      Akku         = unbenutzt
      X-Register   = unbenutzt
      Y-Register   = unbenutzt

jsr FACARG         ; $bc0c
                   ; bzw ARGFAC $bbfc

Ausgabe:
      Akku         = unbenutzt
      X-Register   = unbenutzt
      Y-Register   = unbenutzt
      FAC          = Wert
      ARG          = Wert
```

Der Programmablauf: Gehen wir davon aus, daß Sie die Fläche Ihres Grundbesitzes errechnen wollen. An einer Lagerhalle mit einer Fläche von 2000 m² sind Sie zu einem Drittel beteiligt. Um den Anteil zu ermitteln, müssen wir die Divisionsroutine DIV ($BB12) verwenden.

DIV teilt ARG durch FAC und legt das Ergebnis in FAC ab. Vor Aufruf von DIV muß daher 2000 im ARG und drei im ARG abgelegt werden. Im Programmteil »Anteil berechnen« wird 2000 in $62/$63 abgelegt, das X-Register mit der Zahl $90 geladen und ZWEPOS aufgerufen. Wie wir wissen, befindet sich nach dieser Vorbereitung die Zahl 2000 im Fließkommaformat im FAC.

Diese Fließkommazahl wird durch den Aufruf von FACARG nach ARG kopiert, bevor wir die Zahl drei, mit Hilfe der Routine EINPOS ins Fließkommaformat gewandelt, im FAC ablegen. Der folgende Aufruf von DIV teilt ARG (2000) durch FAC (3) und legt das Ergebnis (666,666667) im FAC ab.

ARG durch FAC = FAC

```
Eingabe:
      FAC          = Teiler
      ARG          = Wert
      Akku         = Exponent d.FAC
      X-Register   = unbenutzt
      Y-Register   = unbenutzt

lda $61            ; Exponent laden,
                     wenn nicht schon im Akku vorhanden
jsr DIV            ; $bb12

Ausgabe:
      Akku         = Eponent d.FAC
      X-Register   = unbenutzt
      Y-Register   = unbenutzt
      FAC          = Ergebnis
      ARG          = Wert
```

Zur Kontrolle wollen wir dieses Ergebnis auf dem Bildschirm mit FACSTR und STROUT ausgeben. Da das Ergebnis in den folgenden Rechnungen gebraucht wird, FACSTR den Inhalt des FAC jedoch zerstört, rufen wir zuvor ein weiteres Mal FACARG auf, um den FAC-Inhalt nach ARG zu kopieren. Nachdem das Ergebnis ausgegeben wurde, wird der Akku mit 13 geladen und BSOUT aufgerufen. BSOUT ist eine Betriebssystemroutine, die das im Akku übergebene Zeichen auf dem Bildschirm ausgibt. Die Einsprungadresse $FFD2 ist dank der erwähnten Sprungtabelle der Betriebssystemroutinen für alle drei Computer identisch.

13 ist der Code für »Carriage Return«. Die Ausgabe dieses Zeichens bewirkt einen Zeilenvorschub und sorgt damit für die optische Trennung der verschiedenen Ergebnisse, die »Grundrechnen« berechnet und ausgibt.

Eine weitere Besonderheit des Ausgabeunterprogramms ist der Aufruf von zwei weiteren Unterprogrammen, die FAC nach PUFFER kopieren beziehungsweise PUFFER nach beendeter Ausgabe wieder nach FAC kopieren. Der Grund ist die erwähnte Zerstörung des Inhaltes von FAC bei Aufruf von FACSTR.

Der nächste Programmteil verwendet die Additionsroutine ADD ($B86A), die FAC und ARG addiert, wobei das Ergebnis wiederum in FAC abgelegt wird. Gehen wir davon aus, daß Ihr Grundbesitz außer einem Drittelanteil an HAUS1 aus einem weiteren Haus mit einer Fläche von 5286 m² besteht, das Ihnen allein gehört. Diese Fläche soll zum vorigen Ergebnis addiert werden.

ARG plus FAC = FAC

```
Eingabe:
      FAC          = Wert 1
      ARG          = Wert 2
      Akku         = Exponent d. FAC
      X-Register   = unbenutzt
      Y-Register   = unbenutzt

lda $61            ; Exponent laden, wenn nicht schon
                     im Akku vorhanden
jsr ADD            ; $b86a

      Ausgabe:
      Akku         = Eponent d. FAC
      X-Register   = unbenutzt
      Y-Register   = unbenutzt
      FAC          = Ergebnis
      ARG          = Wert
```

Zuerst wird der Inhalt von FAC mit FACARG nach ARG kopiert. Anschließend wird mit ZWEPOS die Zwei-Byte-Integerzahl 5286 im Fließkommaformat in FAC abgelegt. Der folgende Aufruf von ADD addiert FAC und ARG. Das Ergebnis wird mit der Ausgaberoutine auf dem Bildschirm ausgegeben (5952,66667), so daß Sie es problemlos mit einem Taschenrechner überprüfen können.

Etwas spät fällt Ihnen nun ein, daß die Grundfläche des zweiten Hauses vor kurzem neu berechnet wurde und das Ergebnis um 52 m² niedriger ausfiel als Ihr alter Wert von 5286 m². Das bisherige Ergebnis muß daher korrigiert werden, Sie müssen 52 subtrahieren.

Mit den Arithmetikroutinen stellt diese Korrektur kein Problem dar. Wir verwenden die Routine SUB ($B853), die FAC von ARG subtrahiert.

Im dritten Programmteil wird zuerst das bisherige Ergebnis mit FACARG wieder in ARG kopiert. Der Routine EINPOS wird die Zahl 52 übergeben, die dadurch als Fließkommazahl in FAC abgelegt wird. EINPOS kann verwendet werden, da es sich bei 52 um einen Ein-Byte-Wert handelt. Wenn Sie in eigenen Programmen nicht von vornherein wissen, wie groß die zu wandelnden Integerzahlen sind, verwenden Sie am besten immer die Routine ZWEPOS.

Nach der Wandlung wird die Routine SUB aufgerufen, die FAC=ARG-FAC rechnet, in unserem Fall 5952,66667-52. Das Ergebnis 5900.66667 wird anschließend ausgegeben.

ARG minus FAC = FAC

Eingabe:

```
     FAC         = Subtrahent
     ARG         = Wert
     Akku        = Exponent d. FAC
     X-Register  = unbenutzt
     Y-Register  = unbenutzt

lda $61          ; Exponent laden, wenn nicht schon
                   im Akku vorhanden
jsr SUB          ; $b853
```

Ausgabe:

```
     Akku        = Exponent d. FAC
     X-Register  = unbenutzt
     Y-Register  = unbenutzt
     FAC         = Ergebnis
     ARG         = Wert
```

Sie kennen nun die Gesamtfläche der Ihnen gehörenden Grundstücke und wollen deren Wert berechnen. Sie gehen dabei von einem durchschnittlichen Quadratmeterpreis von 228 Mark aus. Die Fläche von 5900,66667 m² muß daher mit 228 multipliziert werden.

Die Fläche, die sich noch in FAC befindet, wird mit FACARG nach ARG kopiert, bevor die Ein-Byte-Integerzahl 228 an EINPOS übergeben wird. Durch Aufruf der Routine MULT ($BA2B) werden FAC und ARG miteinander multipliziert. Als Ausgabe erhalten Sie das stolze Ergebnis von 1345352 Mark.

Sie sehen, die Benutzung der Grundrechenroutinen in Assembler stellt keine besonderen Probleme, wenn beachtet wird, daß die Ausgabe von Zwischenergebnissen mit FACSTR und STROUT den Inhalt von FAC zerstört und dieser daher vor der Ausgabe gerettet werden muß.

EXP ($BF7B)

EXP wurde noch nicht erwähnt. Diese Routine arbeitet ebenfalls mit zwei Fließkommazahlen in FAC und ARG. EXP potenziert den ARG mit dem im FAC enthaltenen Exponenten. (FAC=3 und ARG=2 ergibt FAC=8).

ARG mal FAC = FAC

Eingabe:

```
     FAC         = Multiplikator
     ARG         = Wert
     Akku        = Exponent d. FAC
     X-Register  = unbenutzt
     Y-Register  = unbenutzt

lda $61          ; Exponent laden, wenn nicht schon
                   im Akku vorhanden
jsr MULT         ; $ba2b
```

Ausgabe:

```
     Akku        = Exponent d. FAC
     X-Register  = unbenutzt
     Y-Register  = unbenutzt
     FAC         = Ergebnis
     ARG         = Wert
```

ARG hoch FAC = FAC

Eingabe:

```
     FAC         = Exponent
     ARG         = Wert
     Akku        = Exponent d. FAC
     X-Register  = unbenutzt
     Y-Register  = unbenutzt

lda $61          ; Exponent laden, wenn nicht schon
                   im Akku vorhanden
jsr EXP          ; $bf7b
```


Ausgabe:

```
     Akku        = Exponent d. FAC
     X-Register  = unbenutzt
     Y-Register  = unbenutzt
     FAC         = Ergebnis
     ARG         = Wert
```

Die Funktionen

Zur komfortablen Benutzung der Arithmetikfunktionen benötigen wir außer den Grundrechenarten die eingebauten Fließkommafunktionen. Das Programm »Funktionen« (Listing 7) demonstriert den Gebrauch der Funktionen SIN(X), COS(X), SQR(X), LOG(X) und EXP(X). Vor dem Aufruf einer dieser Funktionen muß sich das Argument X der Funktion im FAC befinden. Das Ergebnis des Aufrufs befindet sich anschließend wieder im FAC.

Das Programm »Funktionen« verwendet als Argument die Zahl zehn. Diese Zahl wird der Routine EINPOS im Y-Register übergeben. Nach dem Aufruf von EINPOS befindet sich die Fließkommazahl zehn im FAC und die gewünschte Funktion wird aufgerufen. Das Ergebnis des Funktionsaufrufs wird wie gewohnt mit FACSTR und STROUT auf dem Bildschirm ausgegeben.

```
COS(10) =      -0,544021111  (Adresse: $E264)
SIN(10) =      -0,839071529  (Adresse: $E26B)
SQR(10) =       3,16227766   (Adresse: $BF71)
LOG(10) =       2,30258509   (Adresse: $B9EA)
EXP(10) =  22026,4658        (Adresse: $BFED)
```

Wenn Sie sich davon überzeugen wollen, daß die ausgegebenen Ergebnisse korrekt sind, so können Sie im Direktmodus eingeben:

```
PRINT SIN(10):PRINT COS(10):PRINT SQR(10):
PRINT LOG(10):PRINT EXP(10)
```

und die Ergebnisse mit den Ausgaben des Assemblerprogramms vergleichen.

Funktionen
Eingabe:

FAC	= Wert
Akku	= Exponent d. FAC
X-Register	= unbenutzt
Y-Register	= unbenutzt

```
lda $61          ; Exponent laden, wenn nicht schon
                   im Akku vorhanden
jsr (Funktion)   ;
```

Ausgabe:

Akku	= Exponent d. FAC
X-Register	= unbenutzt
Y-Register	= unbenutzt
FAC	= Ergebnis

Wie Sie sehen, ist der Aufruf einer Funktion noch einfacher als die Anwendung einer Grundrechenroutine. Es genügt, das Funktionsargument vor dem Aufruf in den FAC zu transportieren. Außer den im Programm »Funktionen« verwendeten existiert noch eine Vielzahl weiterer Funktionen, die auf die gleiche Weise auf ein beliebiges Argument X angewendet werden können (TAN, RND etc.).

Sollten Sie mit der Wirkungsweise einiger Funktionen nicht vertraut sein, empfehle ich Ihnen das Studium des Handbuchs, in dem diese bei der Basic-Programmierung beschrieben werden. Denken Sie daran, daß diese Funktionen im Basic-Interpreter integriert sind, daß wir von Assembler aus keine neuen, sondern nur die auch von Basic aus nutzbaren Funktionen ansprechen.

Schnittstellen zu Basic

Bisher wurde die Anwendung der Rechenroutinen und Funktionen in reinen Assemblerprogrammen demonstriert. Eine Hauptanwendung der Maschinenprogrammierung ist jedoch oftmals der Ersatz zeitkritischer Basic-Programmteile durch Assemblerroutinen. Bisher wissen wir jedoch noch nicht, wie von Basic aus Parameter an ein Maschinenprogramm übergeben werden können.



Eine Methode zur Übergabe dürfte jedem von Ihnen bekannt sein: Der Befehl POKE. Mit POKE können wir beliebige Integerzahlen übergeben. Am einfachsten ist die Übergabe einer Ein-Byte-Integerzahl, zum Beispiel durch POKE 250,10. Um mit diesem Befehl einen Zwei-Byte-Wert zu übergeben, kann dieser in ein Low- und ein High-Byte aufgesplittet werden:

```
100 rem *unterprogramm zur parameteruebergabe*
110 rem parameter wird in 'wert' uebergeben
120 hb=int(wert/256) : rem high-byte
130 lb=wert-hb*256 : rem low-byte
140 poke 250,lb:poke 251,hb : rem uebergabe
150 return
```

Dieses Unterprogramm kann vom Hauptprogramm jederzeit mit »GOSUB 120« aufgerufen werden, wenn der zu übergebende Parameter zuvor der Variablen »wert« zugewiesen wurde.

Beispiel:

```
10 wert=1000 : gosub 120
```

Dieser Aufruf teilt die Zahl 1000 in das High-Byte drei und das Low-Byte 232. Das Low-Byte wird in Speicherstelle 250 und das High-Byte in 251 übergeben. Der Maschinenroutine wird der Zwei-Byte-Wert in der gewohnten Form Low-Byte/High-Byte übergeben.

Das Schlüsselloch

Diese Routine sollten Sie jedoch schnellstens wieder vergessen. Die vorgestellte Methode hat gleich mehrere Nachteile. Zum einen ist es schlichtweg unsinnig, zur Übergabe eines Parameters an eine Maschinenroutine, die einen zeitkritischen Programmteil beschleunigen soll, ein aufwendiges Basic-Unterprogramm zu verwenden. Die Rechnungen und der Unterprogrammaufruf benötigen soviel Zeit, daß der Zeitvorteil der Maschinenroutine wahrscheinlich zumindest teilweise dadurch wieder zunichte gemacht wird, wie im folgenden Beispiel:

```
10 for i=1 to 1000
20   wert=i      : wertzuweisung
30   gosub 120   : uebergabe des wertes
40   sys 49152   : aufruf der maschinenroutine
40 next
```

In diesem Programm werden einer Maschinenroutine nacheinander die Werte eins bis 1000 in einer Schleife übergeben. Das heißt, daß der Basic-Interpreter 1000mal das komplette Basic-Unterprogramm durchlaufen muß!

Der zweite Nachteil unserer Übergaberoutine ist noch schwerwiegender. Mit der vorgestellten Routine ist es völlig unmöglich, Fließkommazahlen an ein Maschinenprogramm zu übergeben. (Außer, Sie haben vor, eine Basic-Übergaberoutine zu schreiben, die eine beliebige Zahl ins Fließkommaformat wandelt und mit POKEs übergibt. Vom Zeitaufwand her gesehen ist es dann jedoch wirklich angebrachter, die Berechnungen in »reinem« Basic durchzuführen. Selbst aufwendige Rechnungen können nur schwerlich langsamer als ein solches Konvertierungsprogramm sein.)

Die USR-Funktion

Basic besitzt eine Schnittstelle zu Maschinenprogrammen, die hervorragend zur Zahlenübergabe, insbesondere zur Übergabe reeller Zahlen (also nicht nur Integerzahlen), geeignet ist, die Funktion USR(X). Dieser Befehl, der im Handbuch des C 64 nur unzureichend erläutert ist, legt eine beliebige Zahl X im Fließkommaformat im FAC ab.

Vor der Verwendung der Funktion USR muß der sogenannte USR-Vektor »verbogen« werden. Dieser Vektor besteht aus zwei Speicherstellen (785/786 beim C 64), in die in der bekannten Form Low-Byte/High-Byte die Startadresse der Maschinenroutine gePOKEt wird.

Die Anwendung der Funktion USR demonstriert das Programm »USR-Vektor« (Listing 8). Das Programm besteht aus zwei Teilen. Teil 1 beginnt ab Adresse $C000 (dezimal 49152), Teil 2 ab Adresse $C100 (dezimal 49408). Der erste Programmteil hat die Aufgabe, die Wurzel einer beliebigen mit USR übergebenen Zahl zu berechnen.

Vor dem Aufruf dieses Programmteils muß der USR-Vektor auf die Startadresse $C000 gerichtet werden. In Speicherstelle 785 wird das Low- und in 786 das High-Byte der Startadresse gepokt. Low- und High-Byte des dezimalen Wertes 49152 können in Basic mit Hilfe der beschriebenen Routine (Zeile 120-130) ermittelt werden. In unserem Fall ergibt sich als Low-Byte der Wert null und als High-Byte die 192. Der USR-Vektor wird daher mit folgenden Befehlen auf den ersten Programmteil gerichtet:

```
POKE 785,0:POKE 786,192
```

Geben Sie diese Befehle nach dem Assemblieren des Programms bitte im Direktmodus ein. Um die Wurzel von 20 zu berechnen, geben Sie anschließend ebenfalls im Direktmodus ein:

```
PRINT USR(20)
```

Auf dem Bildschirm wird als Ergebnis die Zahl 4,47213595 ausgegeben. Nach dem Aufruf der Funktion USR wurde der übergebene Parameter 20 ins Fließkommaformat gewandelt im FAC abgelegt. Anschließend erfolgte ein indirekter Sprung über den USR-Vektor (JMP(785)), der auf unseren ersten Programmteil zeigt. Dieser indirekte Sprung kann mit einem »SYS 49152« verglichen werden, das heißt unsere Routine wurde gestartet.

Das Ei des Kolumbus

Die Routine ruft die SQR-Funktion auf. Das Argument X ist bereits im FAC vorhanden. SQR berechnet die Wurzel von X und legt sie ebenfalls im Fließkommaformat wieder im FAC ab. Da das Ergebnis im zweiten Programmteil benötigt wird, kopiert die Routine FACPUF den Inhalt des FAC in den Puffer am Programmende, bevor die Rückkehr nach Basic mit RTS erfolgt. Der Basic-Befehl PRINT führt nun zur Ausgabe der im FAC enthaltenen Fließkommazahl, das heißt der Wurzel von 20.

USR übergibt somit eine Zahl nach Wandlung ins Fließkommaformat im FAC an ein Maschinenprogramm, das anschließend über den USR-Vektor aufgerufen wird und das Ergebnis von Rechenoperationen ebenfalls im FAC an Basic zurückübergibt.

Das im FAC übergebene Ergebnis kann von Basic beliebig weiterverarbeitet, zum Beispiel einer Variablen zugewiesen werden:

```
A=USR(1000):PRINT A
```

Die Eingabe dieser Befehle im Direktmodus führt zur Ausgabe von 31,6227766. Das Ergebnis der Rechenoperationen wurde der Variablen A zugewiesen.

Bevor wir uns dem zweiten Programmteil zuwenden, geben Sie bitte noch ein letztes Mal ein:

```
PRINT USR(20)
```

damit unser erster Programmteil die Wurzel von 20 in den Puffer kopiert. Der zweite Programmteil soll eine ebenfalls mit USR zu übergebende Zahl mit dieser Wurzel multiplizieren.

Da sich Teil 2 ab Adresse $C100 im Speicher befindet, muß der USR-Vektor auf diese neue Startadresse gerichtet werden:

```
POKE 785,0:POKE 786,193
```

Geben Sie ebenfalls im Direktmodus ein:

```
PRINT USR(5.3)
```

Als Ausgabe erhalten wir 23,7023206, was exakt der Multiplikation von 5,3 mit der zuvor errechneten Wurzel von 20 entspricht. Dem zweiten Programmteil wurde nach dem Aufruf mit USR die Zahl 5,3 im FAC übergeben. Diese Zahl wird durch FACARG nach ARG kopiert, bevor der Inhalt des Puffers, die von Teil 1 berechnete Wurzel von 20, nach FAC kopiert wird. Der Aufruf von MULT führt zur Multiplikation von FAC mit ARG und Ablage des Ergebnisses im FAC. Dieses Ergebnis wird mit dem PRINT-Befehl ausgegeben.

USR mit mehreren Parametern

Nachdem die prinzipielle Funktionsweise des Befehls USR nun deutlich wurde, stelle ich eine von vielen denkbaren Methoden vor, beliebig viele Parameter mit USR an ein Maschinenprogramm zu übergeben.

Das Programm »USR-Vektor.2« (Listing 9) verwendet folgende Methode: Das aufrufende Basic-Programm übergibt mit POKE in Speicherzelle 250 die Anzahl der Übergabeparameter. Das Maschinenprogramm kopiert nach jedem Aufruf mit USR die im FAC übergebene Fließkommazahl in den Puffer am Programmende. Um ein Überschreiben zu vermeiden, muß jede weitere Fließkommazahl fünf Byte hinter der zuletzt kopierten in den Puffer geschrieben werden.

In diesem Fall ist die Programmierung einfacher als die eines sich selbst verändernden Programms.

Jedesmal, wenn eine Fließkommazahl in den Puffer kopiert wurde, wird der Befehl STA PUFFER,X verändert. Die Adresse PUFFER wird um den Wert fünf inkrementiert, so daß sie unmittelbar hinter die zuletzt abgespeicherten Daten zeigt. Bei jeder Parameterübernahme wird die Speicherzelle 250 dekrementiert. Der Wert null zeigt an, daß alle zu übergebenden Parameter in den Puffer kopiert wurden und der Programmteil »Rechnen« wird angesprungen, der die Aufgabe besitzt, alle übernommenen Zahlen zu addieren.

»Rechnen« verwendet ebenfalls diese Programmiertechnik. Der Inhalt von FAC (die letzte übergebene Zahl) wird mit FACARG nach ARG kopiert, der erste übergebene Parameter aus dem Puffer nach FAC kopiert und ADD angesprungen. Anschließend wird die Adresse im Befehl LDA PUFFER,X um fünf inkrementiert, so daß sie auf die nächste Fließkommazahl im Puffer weist.

Bevor diese zur bisherigen Summe addiert wird, muß überprüft werden, ob die Adresse im Befehl LDA PUFFER,X bereits soweit erhöht wurde, daß sie mit der Adresse im Befehl STA PUFFER,X identisch ist. Wenn ja, wurde die letzte übergebene Fließkommazahl bereits behandelt und die Aufsummierung wird durch die Rückkehr nach Basic beendet. Im FAC befindet sich nun die Summe aller übergebenen Zahlen.

Wenn Sie dieses Programm assembliert haben, geben Sie bitte folgendes Programm ein:

```
100 poke 785,0:poke 786,192
:rem usr-vektor auf routine
110 an=100:rem anzahl aufzusummierender elemente
120 poke 250,an:rem anzahl in 250 uebergeben
130 for i=1 to an:s=usr(i):next:rem parameter
140 print s:rem ergebnis der summierung
```

Erklärungsbedürftig in diesem Programm ist die Zeile 130. In der Schleife werden der Maschinenroutine der Reihe nach die Zahlen eins bis 100 übergeben. Nach jedem Aufruf mit USR(I) wird der gerade übergebene Wert der Variablen S zugewiesen. Nach der letzten Übergabe und der darauf folgenden Ausführung des Programmteils »Rechnen« befindet sich im FAC die Summe, die nun mit S=USR(I) der Variablen S zugewiesen wird. Der Befehl PRINT S führt zur Ausgabe der Summe 5050, die mit den Basic-Befehlen:

```
for i=1 to 100:x=x+i:next:print x
```

überprüft werden kann. Ich hoffe, »Puritaner« mögen mir die verwendete Programmiertechnik verzeihen. Dieses Programm kann selbstverständlich auch unter Verwendung der indirekt indizierten Adressierung geschrieben werden »LDA

(POINTER),Y«. Was demonstriert werden sollte: daß mit dem USR-Befehl prinzipiell die Übergabe beliebig vieler reeller Zahlen an eine Maschinenroutine möglich ist (wobei die maximale Parameteranzahl bei vorliegendem Programm etwa 800 beträgt, da ansonsten der Puffer durch Überschreitung der Adresse $CFFF »überläuft«).

Welche Rechenoperationen Sie mit den übergebenen Parametern ausführen, ob Sie sie aufsummieren, multiplizieren oder statistische Berechnungen wie zum Beispiel die Berechnung von Mittelwert oder Streuung durchführen, bleibt völlig Ihnen überlassen.

CHKKOM, GETBYT, FRMNUM, ADRFOR, GETADR

Wie wir sahen, können mit USR problemlos »krumme« Werte an eine Maschinenroutine übergeben werden. Wir sahen ebenfalls, daß die Übergabe mehrerer Parameter erheblich aufwendiger ist. Der Basic-Interpreter stellt uns jedoch noch weitere Routinen zur Verfügung, mit denen eine unkomplizierte Übergabe beliebig vieler Parameter möglich ist.

Es sind Routinen, die keine »offizielle« Schnittstelle wie die USR-Funktion darstellen, sondern die der Basic-Interpreter selbst bei der Abarbeitung eines Programms benötigt. Mit diesen Routinen ist es möglich, Parameter direkt aus dem Basic-Text zu lesen, die hinter einem SYS-Befehl aufgereiht übergeben werden, zum Beispiel mit SYS 49152,10,20,30.

Zuerst müssen wir wissen, daß der Basic-Interpreter in den Speicherstellen 122/123 (C 64) einen Pointer auf den Basic-Text verwaltet, der auf das aktuelle Zeichen weist. Nach Abarbeitung des SYS-Befehls, mit dem unsere Routine aufgerufen wird, weist dieser Pointer auf das der Adresse (49152) folgende Komma. Da hinter dem Komma der erste Übergabeparameter folgt, benötigen wir eine Routine, die es ermöglicht, das Komma einzulesen und den Textpointer auf den ersten Parameter zu setzen.

Zum Einlesen eines Kommas wird die Routine CHKKOM ($AEFD) verwendet. Durch Aufruf von CHKKOM wird das aktuelle Zeichen eingelesen, auf das der Textpointer zeigt und überprüft, ob es sich bei diesem Zeichen um ein Komma handelt. Fällt der Test negativ aus, wird ein »Syntax error« ausgegeben. War das eingelesene Zeichen tatsächlich ein Komma, wird der Textpointer inkrementiert – und weist somit auf das nächste einzulesende Zeichen –, bevor das Unterprogramm CHKKOM mit RTS beendet wird.



Dank CHKKOM können wir mehrere Parameter durch ein definiertes Zeichen voneinander unterscheiden und überprüfen, ob der Benutzer unserer Routine die verschiedenen Parameter korrekt mit Kommatas getrennt hat. Das Einlesen und die Überprüfung des Trennzeichens nützt uns jedoch wenig, solange wir nicht an die eigentlichen Parameter herankommen.

Die einfachste Routine zum Einlesen eines Parameters nennt sich GETBYT ($B79E). Vor dem Aufruf von GETBYT muß der Textpointer auf das erste Zeichen des Parameters weisen. Da der Aufruf von CHKKOM den Textpointer inkrementiert und dieser damit auf das erste Zeichen hinter (!) dem Komma weist, ist diese Bedingung erfüllt. Eine zweite Bedingung besteht darin, daß der übergebene Parameter ein Ein-Byte-Wert sein muß, entweder eine Ein-Byte-Zahl (null bis 255) oder aber eine Variable, der ein Ein-Byte-Wert zugewiesen wurde (10 a=200:sys 49152,a), da sonst ein »syntax error« ausgegeben wird. Nach dem Einlesen des Parameters weist der Textpointer übrigens ebenso wie bei den noch zu besprechenden Routinen FRMNUM und GETPOS hinter das letzte Zeichen der Zahl beziehungsweise Variablen, bei der Übergabe mehrerer durch Kommatas getrennter Variablen somit auf das nächste Komma.

SYS mit mehreren Parametern

GETBYT übergibt den eingelesenen Parameter im X-Register. Mit CHKKOM und GETBYT können beliebig viele Ein-Byte-Werte übergeben werden, wie das folgende Beispiel zeigt:

```
jsr CHKKOM     ;komma lesen
jsr GETBYT     ;1. parameter in x einlesen
stx PUFFER     ;1. parameter nach puffer
jsr CHKKOM     ;komma lesen
jsr GETBYT     ;2. parameter in x einlesen
stx puffer+1   ;2. parameter nach puffer+1
jsr CHKKOM     ;komma lesen
jsr GETBYT     ;3. parameter in x einlesen
stx puffer+2   ;3. parameter nach puffer+2
```

Diese Routine liest drei einem SYS-Befehl folgende und durch Kommata getrennte Ein-Byte-Werte ein, die in Puffer, Puffer+1 und Puffer+2 abgelegt werden. Die Parameter können sowohl direkt angegebene Zahlen als auch Variablen sein, deren Inhalt GETBYT ermittelt:

1. sys 49152,10,20,30
2. a=10:b=20:c=30:sys 49152,a,b,c
3. b=20:sys 49152,10,b,30

Die Routine FRMNUM ($AD8A) ähnelt der USR-Funktion. FRMNUM liest einen beliebigen numerischen Ausdruck und wertet ihn aus. Das Ergebnis wird im Fließkommaformat im FAC hinterlegt. Außer als Ersatz der USR-Funktion kann FRMNUM in Verbindung mit der Routine ADRFOR ($B7F7) zur einfachen Übergabe von Zwei-Byte-Integerwerten verwendet werden. ADRFOR wandelt die Fließkommazahl im FAC in eine Zwei-Byte-Integerzahl, die im Akku und Y-Register übergeben wird (Y=Low-Byte/Akku=High-Byte).

Mit CHKKOM, GETBYT, FRMNUM und ADRFOR können nach einem SYS-Befehl Ein- oder Zwei-Byte-Werte und auch reelle Zahlen übergeben werden. Das Einlesen eines Zwei-Byte-Wertes erfordert den Aufruf von CHKKOM, FRMNUM und ADRFOR:

```
jsr CHKKOM
jsr FRNUM
jsr ADRFOR
```

Aufruf zum Beispiel mit: sys 49152,1000

Leider nur für den C128 bekannt ist mir die Adresse der Routine GETADR ($880F), die diese drei Routinen nacheinander aufruft. Beim C128 können Sie sich somit zum Einle-

sen einer Zwei-Byte-Zahl in Akku und Y-Register auf den Aufruf von GETADR beschränken.

Eine weitere interessante Routine ist ADRBYT ($B7EB). Diese Routine ruft nacheinander FRMNUM, ADRFOR, CHKKOM und GETBYT auf, liest also zuerst einen Zwei-Byte-Wert, danach das Komma und anschließend einen Ein-Byte-Wert ein, was zum Beispiel für Grafikroutinen interessant sein kann (X-Richtung: 0..319; Y-Richtung: 0..199). Da Sie sich nun sicherlich fragen, ob der ADRFOR folgenden Aufruf von CHKKOM nicht mit den im Akku und Y-Register übergebenen Zwei-Byte-Wert überschreibt: der Wert wird tatsächlich überschrieben. ADRFOR übergibt den Parameter jedoch nicht nur in den beiden Prozessor-Registern, sondern legt ihn zusätzlich noch in den Speicherstellen $14/$15 ab.

Formeln sind erlaubt

Das Programm »Parameter lesen« (Listing 10) demonstriert die Anwendung der beschriebenen Routinen. Geben Sie bitte nach der Assemblierung folgendes Basic-Programm ein:

```
100 SYS 49152,10,2000,5000,60,2*3.4
```

Der Reihe nach werden folgende Zahlen ausgegeben werden:

```
10
2000
60
5000
 6,8
```

Nachdem mit CHKKOM das erste Komma gelesen wurde, wird mit GETBYT die Zahl zehn ins X-Register gelesen. Da der Akku keinen definierten Wert besitzt, wird er mit null geladen, bevor die übergebene Zahl mit INTOUT ausgegeben wird. Erinnern Sie sich daran, daß INTOUT eine Zwei-Byte-Integerzahl ausgibt (X=Low-Byte/Akku=High-Byte).

Das nächste Komma wird eingelesen und danach mit FRMNUM und ADRFOR die Zwei-Byte-Zahl 2000. Da ADRFOR das Low-Byte im Y-Register und das High-Byte im Akku übergibt, die Routine INTOUT das Low-Byte jedoch im X-Register erwartet, wird der Inhalt von Y ins X-Register kopiert.

Mit CHKKOM und ADRBYT werden in einem Zug die Zahl 5000 und die Zahl 60 gelesen, die sich nach ADRBYT im X-Register befindet und mit INTOUT ausgegeben wird (zuvor wird wieder das High-Byte gelöscht, das heißt der Akku mit null geladen).

Auf die zuerst eingelesene Zahl 5000 kann wie erläutert über die Speicherstellen $14/$15 zugegriffen werden. Nachdem sowohl X-Register als auch Akku mit den entsprechenden Inhalten geladen wurden, wird INTOUT aufgerufen.

Der letzte Programmteil zeigt, daß mit FRMNUM beliebige numerische Ausdrücke sehr einfach im Fließkommaformat an den FAC übergeben werden können. Meiner Ansicht nach ist FRMNUM der Funktion USR vorzuziehen, da es nicht in jedem Fall erwünscht ist, daß nach dem Aufruf von USR unweigerlich ein Sprung über den USR-Vektor erfolgt.

Sie besitzen nun alle Mittel, die Sie benötigen, um beliebige Parameter von Basic an eine Maschinenroutine zu übergeben, oder?

Variablen suchen

Nun, eine wesentliche Möglichkeit der Parameterübergabe wurde bisher übergangen: die Übergabe der Strings! Wir können zwar beliebige numerische Werte übergeben, besitzen jedoch vorläufig keine Ahnung, wie ein Maschinenprogramm auch auf Strings zugreifen kann.

Der Zugriff auf Strings ist eigentlich in einem Artikel über Arithmetikroutinen fehl am Platz. Ich erwähne ihn dennoch, weil wir im folgenden die flexibelste Routine zur Parameterübergabe besprechen werden und diese sich sowohl auf numerische Variablen als auch auf Strings bezieht.

GETPOS

GETPOS ($B08B) ist eine der wichtigsten Routinen des Basic-Interpreters. Assemblerunterprogramme für ein Basic-Programm zu schreiben, wird erst mit GETPOS wirklich komfortabel.

Der Routine GETPOS wird ein beliebiger (!) Variablenname übergeben. Diese Routine sucht die zugehörige Variable und übergibt im Akku und Y-Register einen Zeiger auf die Variable. Im Akku befindet sich das Low- und im Y-Register das High-Byte dieses Zeigers.

Um GETPOS verwenden zu können, müssen Sie jedoch wissen, wie der Basic-Interpreter Variablen ablegt:

1. Integer: (Name)(Name)(High)(Low)(.)(.)(.)(Name...
2. Real: (Name)(Name)(5 Byte Fließkomma)(Name...
3. String: (Name)(Name)(Länge)(Adr-Low) Adr-High)(.)(.)(Name...)

Unabhängig vom Variablentyp werden bei nichtdimensionierten Variablen sieben Byte pro Eintrag in die sogenannte »Variablentabelle« verwendet. Die beiden ersten Bytes sind immer die ersten signifikanten Zeichen des jeweiligen Variablennamens.

Bei Integervariablen folgt der Wert in der ungewohnten Form Low-Byte/High-Byte und die nächsten drei Byte sind ungenutzt. Bei Realvariablen besteht die Zahl selbst aus den Bytes drei bis sieben (Fließkommaformat). Für Strings enthält die Variablentabelle die sogenannten »Stringdescriptoren«, nicht den String selbst. Die Stringdescriptoren bestehen aus einem Byte für die Stringlänge und zwei Byte für die Adresse (im gewohnten Format Low/High), an der sich der String selbst befindet, das heißt einen Zeiger auf den String. Die beiden nächsten Bytes sind ebenso wie bei Integervariablen ungenutzt.

Dimensionierte Felder

Folgende Änderungen müssen bei dimensionierten Variablen beachtet werden: Der Name der Arrayvariablen befindet sich nur einmal am Arrayanfang (zwei Byte). Die folgenden beiden Bytes geben den vom Array belegten Speicherplatz an, ein weiteres Byte die Arraydimension, und Byte sechs und sieben enthalten die Anzahl der Arrayelemente. Nach dieser Beschreibung folgen die einzelnen Elemente des Arrays, wobei jedoch speicherplatzsparender als bei nichtdimensionierten Variablen vorgegangen wird:

1. Integer: (Var1:Low)(Var1:High)(Var2:Low)(Var2:High)(Var...
2. Real: (Var1: 5 Byte)(Var2: 5 Byte)(Var3: 5 Byte)(Var...
3. String (Var1:Länge)(Var1:Adr-Low) (Var1:Adr-High)(Var2:Länge..

Wie Sie sehen, sind dimensionierte Arrays weit kompakter aufgebaut als nicht dimensionierte. Die Variablentabelle selbst befindet sich beim C16 und C64 unmittelbar hinter dem Ende des Basic-Programms, beim C128 ab $0400 in Bank 1, der Variablenbank.

Die Strings werden bei allen drei Computern ab dem Ende des RAM-Speichers abwärts angelegt (in Bank 1 beim C128). In diesem »String-Stack« liegen die Strings im ASCII-Format abgelegt unmittelbar hintereinander. Beim C128 existiert zusätzlich zu jedem String ein »Rückwärtszeiger«, der auf die zu diesem String gehörenden Descriptoren weist und die Garbage Collection erheblich beschleunigt:

C16/C64:

MaierMüllerWiedemann...

C128:
Maier(Pointer)Müller(Pointer)Wiedemann(Pointer)...

Der Aufruf von GETPOS liefert immer (!) im Akku und Y-Register einen Pointer (Zeiger) auf das erste Byte des Descriptors der jeweiligen Variablen, unabhängig vom Variablentyp und unabhängig davon, ob es sich um eine dimensionierte oder einfache Variable handelt. Der übergebene Pointer weist daher bei nichtdimensionierten Variablen auf das erste Byte des Variablennamens und bei dimensionierten Variablen auf das erste Byte der Variablen selbst, beziehungsweise bei dimensionierten Stringvariablen auf den Längendescriptor der Stringvariablen.

Wenn GETPOS feststellt, daß noch keine Variable mit dem übergebenen Namen existiert, wird eine neue Variable unter diesem Namen angelegt.

Für die Übergabe aus Basic können die verschiedensten Formate verwendet werden:

1. SYS 49152,I% SYS 49152,I%(2) SYS 49152,I%(J)
2. SYS 49152,R SYS 49152,R(2) SYS 49152,R(J)
3. SYS 49152,S$ SYS 49152,S$(2) SYS 49152,S$(J)

Die zugehörige Assemblerroutine (unabhängig vom Variablentyp!):

```
JSR CHKKOM   ;KOMMA LESEN
JSR GETPOS   ;=> POINTER (AKKU=LOW/Y=HIGH)
```

Mit diesen Kenntnissen können wir nun endlich unsere Arithmetikroutinen effektiv einsetzen. Im Demoprogramm »USR-Vektor.2« wurden Realzahlen aufsummiert, wobei jede Zahl mit einem USR-Aufruf an die Maschinenroutine übergeben wurde. Diese Übergabe (Basic-Schleife!) ist jedoch langsam, daß das Basic-Programm »FOR i=1 to 100:I=I+1: NEXT« fast exakt gleich schnell ist, wie Sie jederzeit ausprobieren können.

Diese langwierige Übergabe entfällt jedoch völlig, wenn die betreffenden Zahlen in einem Array abgelegt werden und Basic dem Assemblerprogramm nur noch den Namen des Arrays mitteilen muß, das die Maschinenroutine nun selbständig bearbeiten kann. Um das Programm komfortabler zu gestalten, sehen wir die Möglichkeit vor, nur einen bestimmten Arrayteil aufzusummieren. Das Basic-Programm muß in diesem Fall zwei Parameter übergeben, die mit GETPOS gelesen werden: den Namen des ersten und den Namen des letzten Arrayelementes, das in die Summierung einbezogen werden soll, also die Intervallgrenzen.



Sollen zum Beispiel die ersten hundert Elemente des Arrays A(..) summiert werden, lautet der Aufruf:

```
SYS 49152,A(1),A(100)
```

Das Programm »Summierung« (Listing 11) demonstriert die Geschwindigkeitsvorteile von Assembler ohne die »Bremse« der Basic-Übergabe. Zweimal hintereinander wird CHKKOM und GETPOS aufgerufen. Im ersten Fall wird durch GETPOS ein Pointer auf das erste zu behandelnde Arrayelement A(1) übergeben, der nach START/START+1 kopiert wird, im zweiten Durchgang wird der auf A(100) – die rechte Begrenzung – übergebene Zeiger nach ENDE/ENDE+1 kopiert.

Das Unterprogramm INFAC verwendet den Zeiger START zum Aufruf von KONFAC, jener Routine, die eine Fließkommakonstante nach ARG kopiert. Der erste Aufruf dieses Unterprogramms kopiert das Element A(1) nach FAC.

Igel und Hase

In der folgenden Hauptschleife wird FAC nach ARG kopiert und der Pointer START(+1) um fünf erhöht. Der Pointer weist anschließend auf das erste Byte der Realzahl A(2). Es folgt ein Vergleich von START(+1) mit ENDE(+1). Wenn START(+1) größer ist als ENDE(+1), wurde die übergebene rechte Arraygrenze bereits überschritten, die Arbeit der Routine ist beendet und die Summe, die sich im FAC befindet, wird mit FACSTR und STROUT ausgegeben.

Da START(+1) momentan erst auf das Element A(2) weist, ist unsere Routine jedoch noch nicht fertig mit ihrer Aufgabe. A(2) wird mit INFAC in den FAC kopiert, FAC und ARG addiert (JSR ADD, das Ergebnis befindet sich anschließend in FAC) und zum Schleifenanfang gesprungen.

Der Programmablauf ist damit klar. Interessant ist nun ein Zeitvergleich zwischen der Assembler- und einer äquivalenten Basic-Routine. Zuerst muß jedoch das Array initialisiert werden:

```
100 DIM A(2000)
110 FOR I=1 TO 1000:A(I)=I/2:NEXT
120 PRINT"INITIALISIERUNG BEENDET"
130 TA=TI:SYS 49152,A(1),A(1000):TB=TI:PRINT:PRINT"
 ASS.:"(TB-TA)/60
140 TA=TI:FOR I=1 TO 1000:S=S+A(I):NEXT:TB=TI:
PRINT S:PRINT"BASIC:"(TB-TA)/60
```

Initialisiert wird ein Array mit 2000 Elementen, die ersten 1000 Elemente werden mit den Zahlen: 0,5; 1; 1,5; 2;...; 500 besetzt. In Zeile 130 wird die Assemblerroutine aufgerufen und die Zeit mit der internen Uhr TI gestoppt. Anschließend werden die Elemente A(1) bis A(1000) in Basic addiert und dabei ebenfalls die Zeit gemessen.

Wenn ich ehrlich bin, muß ich zugeben, daß das Ergebnis mich selbst verblüfft hat. Die Fließkommaarithmetik ist eine extrem aufwendige Angelegenheit und ich war bisher der Ansicht, daß Assembler bei Verwendung der Fließkommaroutinen keine allzu großen Vorteile gegenüber Basic besitzt, da sich an der langsamen Arbeitsweise dieser Routinen natürlich nichts ändert, wenn sie statt von Basic von einer Maschinenroutine aus aufgerufen werden. Hier das angesichts dieser Tatsache erstaunliche Ergebnis:

1000 Realzahlen addieren:
Basic: 6,25 sec
Assembler: 0,78 sec

Trotz der »gemütlichen« Fließkommaarithmetik ist die Assemblerroutine somit immerhin um den Faktor acht schneller als das vergleichbare Basic-Programm. Extreme

Ergänzen Sie jetzt Ihre 64'er-Sammlung

Schaffen Sie sich ein interessantes Nachschlagewerk und gleichzeitig ein wertvolles Archiv!

Kennen Sie alle Ausgaben von 64'er? Suchen Sie einen ganz bestimmten Testbericht? Oder haben Sie einen Teil eines interessanten Kurses versäumt? Suchen Sie nach einer speziellen Anwendung?

Damit Sie jetzt fehlende Hefte mit »Ihrem« Artikel nachbestellen können, finden Sie auf diesen Seiten eine Zusammenstellung aller wesentlichen Artikel der Ausgaben 01 bis 12/85.

Und so kommen Sie schnell an die noch lieferbaren Ausgaben: Prüfen Sie, welche Ausgabe in Ihrer Sammlung noch fehlt, oder welches Thema Sie interessiert. Tragen Sie die Nummer dieser Ausgabe und das Erscheinungsjahr (z.B. 2/85) auf dem Bestellabschnitt der hier eingehefteten Bestell-Zahlkarte ein. Die ausgefüllte Zahlkarte einfach heraustrennen und Rechnungsbetrag beim nächsten Postamt einzahlen. Ihre Bestellung wird nach Zahlungseingang umgehend zur Auslieferung gebracht.

Aktuell

Stichwort	Titel	Seite	Ausgabe
Allgemeines	Commodore Gestern Heute Morgen	10	01/85
Computer	Amiga — Der neue Supercomputer	8	09/85
Interview	Interview mit David Crane (Game Designer)	146	06/85
Lernen	Schule braucht Computer (VAM-Computer)	9	06/85
Messen	International Chaos Communication Congress	15	03/85
	Heiße Messe in der Wüste: CES	8	03/85
	Hannover-Messe '85	8	06/85
	Hannover-Messe '85	8	07/85
	Chicago im Zeichen der CES	8	08/85
	Aktuelles von der C'85 in Köln	15	08/85
	Btx Total (Internationale Funkausstellung)	8	10/85
	PCW-Computermesse in London	8	11/85
	Neues von der Commodore-Fachausstellung 1985	8	12/85
Recht	Die neue Abmahnmasche — Vorsicht bei Programmangeboten	8	05/85
	Die Ex-Knacker — wo sind sie geblieben?	27	08/85
	Interview mit Raubkopierern (Section 8)	28	08/85
	Schützer kontra Knacki's	23	08/85
	Raub-Talkshow	12	08/85
	Das Urheberrechtsgesetz und Gedanken zu seiner Anwendung	21	08/85
	Änderung des Urheberrechtsgesetzes	162	09/85

Buchbesprechungen

Stichwort	Titel	Seite	Ausgabe
Anfänger	Goldmann Computer Compact	87	03/85
	Basic-Wegweiser für den C 64	86	05/85
	Alles über den C 64, Sachbuchreihe, Band 1	115	06/85
	Lehrspielzeug Computer: C 64/VC 20	112	11/85
	C 64 Computerhandbuch	171	11/85
	Einführungskurs: Commodore 64	144	12/85
Anwendung	Dienstprogramme VC 20, C 64 und SX	86	05/85
	Spaß an Mathe mit dem Commodore 64	88	07/85
	Mathe für die Oberstufe mit dem C 64	88	07/85
	Mathematische Routinen VC 20, Elektrotechnik/Elektronik	112	11/85
	Commodore 64-Listings, Band 2: Dateiverwaltung, Schule, Hobby	112	11/85
	Das Trainingsbuch zum Datamat	144	12/85
C 128	Bücher zum C 128	22	10/85
DFÜ	Das Mailbox-Jahrbuch: Nutz die Netze	112	11/85
Grafik	Grafik auf dem Commodore 64 (+Fehlert. 9/85)	86	05/85
	Einführung in CAD mit dem Commodore 64	128	06/85
	Grafik & Musik auf dem Commodore 64	88	07/85
	Verschiedene Grafikbücher zum C 64	115	08/85
Programmieren	Von Basic zu Assembler: Das Commodore-Buch, Band 4	115	06/85
	64 Intern	115	06/85
	Das Interface Age System-Handbuch zum C 64	115	06/85
	Das C 64 Buch, Band 5: Simons Basic Leitfaden	144	12/85
	Basicode	144	12/85
	Noch mehr Tips und Tricks zum 64er	144	12/85
Speichern	Das Kassettenbuch zum C 64 und VC 20	87	03/85
	Die Floppy 1541 (M&T)	88	07/85
Spiele	Rombachs C 64 Spielführer	87	03/85
	Commodore 64-Listings, Band 1, Spiele	112	11/85
	35 ausgesuchte Spiele für Ihren Commodore 64	171	1/85

64'er Extra

Stichwort	Titel	Seite	Ausgabe
Prozessor	Befehlssatz des 6502/6510-Prozessors	84	09/85
Grafik	Die Videochip-Register des C 64	92	10/85
Sound	Der SID-Chip, seine Register und Programmierung	92	11/85
Speicher	Die Speicherbelegung des C 64	96	12/85

Abenteuerlösungen

Stichwort	Titel	Seite	Ausgabe
Lösungen	Dallas-Quest Lösung	90	01/85
	Guncho Krill-Enchanter ist gelöst	44	03/85
	Infocom-Geheimnisse gelüftet?	49	05/85
	Des Rätsels Lösung: Amazon	145	06/85
	Activision-Adventures entschleiert (Mindshadow, Tracer Sanction)	36	12/85
	Eureka! — ich hab's!	37	12/85
	Lösungen zu Hitchhiker's Guide und Sorcerer	39	12/85

Spiele-Tests

Stichwort	Titel	Seite	Ausgabe
007	James Bond — A View to a Kill	156	09/85
Abenteuer	Abenteuerpaket 1	48	08/85
	Shadowfire	146	09/85
	The Quest — mit C 64 auf Suche nach Drachen	47	01/85
Action	Hexenküche	50	07/85
	Master of the Lamps	48	07/85
	Rescue on Fractalus	158	10/85
	Stellar 7	49	08/85
Construction Set	Mail Order Monsters	49	08/85
	Racing Destruction Set	50	08/85
Geschicklichkeit	Australopedicus Robustus	50	08/85
	Boulder Dash II	159	10/85
	Crystal Castles	50	07/85
	Gribbly's Day out	148	09/85
	Rock'n Bolt	48	08/85
	Thing on a Spring	159	10/85
	Tom + Zaga	48	01/85
Pseudo-Adventures	Roland's Rat Race	49	08/85
	Fourth Protocol und Frankie g.t.H.	162	11/85
Renner	Die Renner 1985: Meistverkaufte Spiele	34	12/85
Schach	Viermal Schachmatt: Verschiedene Schachprogramme	32	12/85
Simulation	Elite	148	09/85
	Jump Jet	148	09/85
	Super Huey Hubschraubersimulator	49	07/85
Sport	Boxspiele: Frank Bruno's B. + Barry McGuigan Champions. B.	49	12/85
	Handkantenschlag per Joystick: Karateka + Exploding Fist	165	11/85
	Nick Faldo Plays the Open (Golf)	159	10/85
	Rallye Speedway	49	07/85
	Slapshot (Eishockey)	50	07/85
	Summer Games II	146	09/85
	World Series Baseball	49	07/85
Diverses	New York City und Air Support	145	06/85

Hardware-Tips und Bauanleitungen

Stichwort	Titel	Seite	Ausgabe
Audio/Video	Mit 5 Mark zu neuen Dimensionen (Stereoanlage am C 64)	34	08/85
	Ein Monitor ist genug (RGB+Composite an C 128)	16	10/85
C 16	Alte Datasette am C 16	31	04/85
	Älter Joystick am C 16	35	05/85
Eingabegeräte	Der Hexer — Zusatztastatur für den MSE	48	10/85
EPROM	EPROMs im Expansion-Port	46	10/85
	EPROM-Trans — Die Super-Erweite[illegible]	42	10/85
	Das 64'er EPROM-Programmiergerät, Teil 1	44	12/85
Floppy/Datasette	Diskettenlaufwerk 1541 selbst justiert	32	10/85
	Die Datasette streikt nie wieder (Anpassung des Tonkopfs)	34	10/85
IEC-Bus	Auf zu neuen Welten: IEC-Bus im Selbstbau (+Fehlerteufel 10/85)	44	07/85
Joystick	Joystick im Selbstbau	33	03/85
	Dauerfeuer-Adapter	46	08/85
RS232/V.24	Das 30-Mark-Interface (Selbstbau RS232)	29	03/85
	Genau betrachtet: Die RS232/V.24-Schnittstelle	80	05/85
Diverses	Userport-Display	36	05/85
	Reset-Taster für alle Fälle (+Fehlert. 9/85)	130	06/85
	Aus eins mach vier (absturzfreie Betriebssystemumschaltung)	41	07/85

Hardware-Grundlagen

Stichwort	Titel	Seite	Ausgabe
Computer	Was bringt der C 128?	28	11/85
Drucker	Welcher Drucker ist der Richtige? (Grundlagen)	15	05/85
	Hammerwerke — wie funktionieren Typenraddrucker	32	06/85
	Die Alternativen: Thermo-, Tintenstrahldrucker + Plotter	24	07/85
Eingabegeräte	Versteht Sie Ihr Computer? (Wie funktionieren Eingabegeräte)	44	09/85
Floppy	Floppy oder Datasette?	129	06/85
Monitore	Wie funktionieren sie, was ist beim Kauf zu beachten?	16	12/85
	Das Kabel zum Monitor: Welche Normen gibt es?	28	12/85
Peripherie	Grafikeingabegerät: Wie funktionieren sie?	30	08/85

Hardware-Tests

Stichwort	Titel	Seite	Ausgabe
Computer	Generationswechsel: Test C 16	16	01/85
	Erster ausführlicher Test C 128 PC (Teil 1)	16	06/85
	Erster ausführlicher Test C 128, PC (Teil 2)	17	07/85
DFÜ	Marktübersicht Modems & Akustikkoppler	32	07/85
Drucker	Vergleich: Drucker unter 700 Mark (Tests und Marktübersicht)	18	05/85
	Tests und Marktübersicht Typenraddrucker	35	06/85
	Test: Brother EP 44	27	07/85
	Brother TC-600	118	08/85
	Riteman C+	133	09/85
	Panasonic KX-P1091	134	09/85
	Star SG 10C	132	09/85
	Melchers CP-80X — wie hätten Sie's denn gern?	25	10/85
	Geheimtip: Der RFI DP 165	24	10/85
	Epson GX 80 — einer für alle	26	10/85
	MPS 803 — ein Drucker für alle Gelegenheiten?	40	1/85
	Epson JX-80 das vielfarbige Druck-Genie	38	11/85
	Epson FX-85 neue Referenz	42	11/85
	SP 1000 VC — Superstar mit Haken	41	11/85
	Der NEC-P2 — das fernöstliche Wunder	159	12/85
	DMPG9 — eine solide Sache	162	12/85
	Das Doppelleben des Joystick-Ports: 10er-Tastaturen	50	09/85
	Joysticks: Test und Marktübersicht (+Fehlerteufel 12/85)	19	11/85
	Es geht auch anders: Lightpens und Trackballs	22	11/85
EPROMer	Frisch gebrannt ist halb gespeichert (EPROM-Programmiergeräte im Test)	39	07/85
	QuickByte II — das Kraftpaket	14	10/85
Floppy/Datasette	Turbo-Floppies, zweite Generation: Speeddos plus + Prologic DOS	28	10/85
	Das große Rennen: Schnelle Bandlaufwerke	37	10/85
	Professionelle Floppylaufwerke für den C 64 (IEC-Floppies)	30	10/85
	Gut gekauft ist halb gespeichert (Marktübersicht Disketten)	38	10/85
Grafik	Die Videowerkstatt (Digitizer-Test)	32	05/85
	Digitalbilder m.d. C 64: PrintTechnik Digitizer	24	01/85
Interface	Hardware-Interface ganz weich: Test EC 64	23	01/85
	Gute Connections — Übersicht Schnittstellen	21	03/85
	Card/Print +6 — Das Allround-Interface	20	03/85
	Das Wiesemann-Centronics-Interface	18	03/85
	Erst ein IEC-Bus öffnet Tür und Tor (+Fehlert.4/6-85)	24	03/85
Monitore	Marktübersicht: Monochrome Monitore	30	12/85
Musik	Trommelwirbel: Test Digital Drums	45	08/85
	Die Musikhardware zum C 64	17	09/85
Roboter	Roboter selbst gebaut (Fischertechnik)	167	10/85
Scanner	So lernt Ihr Drucker lesen	30	06/85
Speicher	Speichertuning VC 20: Test 64 KByte Karte	26	01/85
Steuern	Flottes Türmchen: MEA-Interface	116	08/85

Kurse

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler ist keine Alchimie, Teil 5	142	01/85
	Assembler ist keine Alchimie, Teil 7	124	03/85
	Assembler ist keine Alchimie, Teil 9	138	05/85
	Assembler ist keine Alchimie, Teil 10	127	07/85
	Assembler ist keine Alchimie, Teil 11	126	08/85
	Assembler ist keine Alchimie, Teil 12	109	09/85
	Assembler ist keine Alchimie, Teil 13 (Schluß)	143	10/85
C 128	Entdeckungsreise duch den C 128	42	12/85
Effektives Programieren	Müllabfuhr im Computer: Garbage Collection, Teil 1	122	01/85
	Finden mit System, eine neuartige Suchmethode, Teil 3	148	03/85
	Sortieren mit dem Computer, Teil 2	159	05/85
	Sortieren mit dem Computer, Teil 3	124	06/85
	Sortieren mit dem Computer, Teil 4	138	08/85
	Sortieren mit dem Computer, Teil 5	124	09/85
	Sortieren mit dem Computer, Teil 6 (Schluß)	150	12/85
Extern	C 64 extern — Der Weg nach draußen, Teil 1	144	08/85
	C 64 extern — Der Weg nach draußen, Teil 2	122	09/85
	C64 extern — Der Weg nach draußen, Teil 3 (Schluß)	129	10/85
Floppy	In die Geheimnisse der Floppy eingetaucht, Teil 4	148	01/85
	In die Geheimnise der Floppy eingetaucht, Teil 5	130	03/85
	In die Geheimnise der Floppy eingetaucht, Teil 6	145	05/85
	In die Geheimnisse der Floppy eingetaucht, Teil 7 (Schluß)	116	06/85
	Directory-Manipulationen I	140	06/85
Floppy	Directory-Manipulationen II	163	10/85
Grafik	Hires 3 — 15 neue Basic-Befehle, Teil 2	136	03/85
	Hires 3 — Grafikkurs-Anwendung, Teil 3 (Schluß)	152	08/85
	Sprites ohne Geheimnisse	40	08/85
	Streifzüge durch die Grafikwelt, Teil 1	106	09/85
	Streifzüge durch die Grafikwelt, Teil 2	149	11/85
Logeleien	Logeleien, Teil 1	143	07/85
	Logeleien, Teil 2	136	08/85
	Logeleien, Teil 3 (Schluß)	115	09/85
Musik	Dem Klang auf der Spur, Teil 2	136	01/85
	Dem Klang auf der Spur, Teil 4	131	04/85
	Dem Klang auf der Spur, Teil 5	152	05/85
	Dem Klang auf der Spur, Teil 7	132	07/85
	Dem Klang auf der Spur, Teil 8	133	08/85
	Dem Klang auf der Spur, Teil 9	126	10/85
	Dem Klang auf der Spur, Teil 10 (Schluß)	157	11/85
Speicher	Memory Map mit Wandervorschlägen, Teil 3	126	01/85
	Memory Map mit Wandervorschlägen, Teil 5	144	03/85
	Memory Map mit Wandervorschlägen, Teil 7	120	06/85
	Memory Map mit Wandervorschlägen, Teil 8	140	07/85
	Memory Map mit Wandervorschlägen, Teil 9	129	08/85
	Memory Map mit Wandervorschlägen, Teil 10	112	09/85
	Memory Map mit Wandervorschlägen, Teil 11	133	10/85
	Memory Map mit Wandervorschlägen, Teil 12	145	11/85
	Memory Map mit Wandervorschlägen, Teil 13	146	12/85
Sprachen	Basic ist out — es lebe Forth	43	01/85
VC 20	Der gläserne VC 20, Teil 4	130	01/85
	Der gläserne VC 20, Teil 6 (Schluß)	155	03/85

Software-Tips

Stichwort	Titel	Seite	Ausgabe
C 128	Erste Fragen und Antworten zum C 128	14	09/85
	Fragen und Antworten zum 128er	20	10/85
	Fragen und Antworten zum 128er	40	12/85
Drucker	Der MPS 802 lernt Deutsch	30	05/85
	Centronics-Interface für jeden Bedarf	78	07/85
Textverarbeitung	Software Corner — professionelle Programme richtig eingesetzt (Vizawrite-Tips)	174	12/85
Tips & Tricks	Autoboot beim C 64	86	03/85
	Verbindungsfreundlich (Parallelschnittstelle des VC 20)	91	03/85
	Undefinierte Opcodes des 6502	84	03/85
	Durch POKEs zum Erfolg (Spiele-POKEs)	83	03/85
	Tips und Erweiterungen zu Hi-Eddi und Simons Basic	88	03/85
	Basic-Befehle im Griff	79	05/85
	Durch POKEs zum Erfolg: Spiele-POKEs	78	06/85
	Formatierte Eingabe	148	06/85
	Hi-Text (Text in Hires)	70	08/85
	Verbotene Variablen	66	09/85
	Verschiedene Routinen für Anfänger und Profis (+Fehlerteufel 12/85)	88	11/85
	Der Trick mit dem Joystick (Joystickabfrage)	24	11/85
	Verschiedene Tips für Anfänger und Fortgeschrittene	106	12/85

Software-Grundlagen

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler? Assembler! (Einführung)	32	01/85
	Assembler-Bedienung leicht gemacht, Teil 1	169	12/85
DFÜ	Der erste Kontakt mit DFÜ	40	06/85
	Die Netze der Post: Btx, Datex-P, Telebox	46	06/85
	DFÜ — Was ist das?	44	06/85
	Mailbox für Anfänger	30	07/85

Stichwort	Titel	Seite	Ausgabe
Datei	Die wichtigsten Begriffe der Dateiverwaltung	42	05/85
	Dateiverwaltung ist nicht gleich Datenbank	44	05/85
	Dateiverwaltung: Was Sie beim Kauf beachten sollten	40	05/85
Drucker	Hardcopy leicht gemacht (wie programmiert man Hardcopies)	34	09/85
EPROM	Wie sage ich es meinem EPROM? (EPROM-Grundlagen)	35	07/85
Funktionen	Funktionen für Anfänger	164	05/85
Lernen	Besser lernen mit dem Computer	166	10/85
Musik	Klangprogrammierung ohne Ballast	19	09/85
Spiele	Taktik- und Strategiespiele	46	03/85
	Play by Mail und Play by Modem	153	09/85
Sprachen	Sprachen für Computer, Teil 2	46	06/85
Textverarbeitung	Von der Schreibmaschine zum Textsystem	34	03/85

Listings zum Abtippen

Stichwort	Titel	Seite	Ausgabe
Anwendung	Der C 64 als Handballtrainer (AdM)	52	01/85
	Ligatab — ohne Organisation kein Tor (LdM)	50	03/85
	Gut Ziel mit dem C 64 — Schützenvereinsergebnisse (AdM)	52	03/85
	Weißt du, wieviel Sternlein stehen (Sternkarte) (AdM) (+ Fehlert. 6/85)	52	05/85
	Haushaltsbuchführung (AdM)	52	07/85
	Netzwerkanalyse: Ein Programm für Hobby-elektroniker (AdM)	52	08/85
	Prüfungsfragen (AdM)	52	09/85
	Fit in Latein mit dem C 64 (AdM)	52	10/85
	Lyrik-Maschine (AdM)	52	11/85
	Hypra-Platos (LdM)	50	11/85
	Der Chemie-Assistent (AdM)	52	12/85
	SMON Teil 3: Ohne gutes Werkz. geht es nicht	69	01/85
	Hypra-Ass (LdM)	51	07/85
	Neues vom SMON (+ Fehlerteufel 11/85)	87	10/85
	Reassembler zu Hypra-Ass (+ Fehlerteufel 12/85)	97	11/85
	Ergänzungen zu Hypra-Ass (bedingte Verzweigungen)	96	11/85
	Tips & Tricks zum SMON (inklusive Diskmonitor)	100	12/85
Bildschirmseite	Auflösung Wettbewerb Bildschirmseite: Drei Top-Programme	158	09/85
DFÜ	Terminalprogramm der Spitzenklasse (+ Fehlerteufel 10/85)	149	07/85
Datei	SMU — Der Maskengenerator (LdM)	50	12/85
Drucker	Hi-Eddi-Druckerroutinen	69	06/85
	C 64 Schreiberling — Drucken wie gemalt	54	10/85
	Koalabilder Farbhardcopy auf Epson JX-80	39	11/85
Einzeiler	Die nächsten 14 aus d. Einzeilerwettbewerb	157	01/85
Floppy	Hypra-Load mal 4 (+ Fehlerteufel 3/85)	82	01/85
	Diskettenmonitor	83	08/85
	Disk-Designer	70	09/85
	Herzoperation (Hypra-Load + Hypra-Ass + DOS5.1 + Centronics)	104	11/85
Grafik	Vier Pseudo-VICs mit 32 Sprites	76	01/85
	Hi-Eddi: Zeichen- und Malproggramm (LdM)	50	01/85
	Elektrotechnisches Zeichnen mit dem VC 20	71	03/85
	Mini-Grafik VC 20, Grafikhilfe	69	05/85
	Trickfilm mit dem C 64: Bewegte 3D-Grafik (LdM) (+ Fehlerteufel 6/85)	51	05/85
	Kurvenplotten mit Hardcopy auf dem C 16	68	06/85
	Doppelte Grafikauflösung für C 128	33	11/85
	Bilder aus einer anderen Dimension (Apfelmännchen)	80	11/85
Intelligenz	VIC — das intelligente Programm (Wettbewerbssieger)	173	05/85
Musik	Sound Machine (+ Fehlerteufel 10/85)	23	09/85
	Sound Master (Basic-Erweiterung)	31	09/85
Spiele	6510 — Die Suche nach der Prozessor	70	05/85
	Samurai (Strategiespiel)	72	06/85
	Schach dem C 64: Schachprogramm zum Abtippen	72	08/85
	Spielen auf zwei Bildschirmen: Zeichensatzscrolling (LdM)	51	09/85
	Pac-Man unter der Lupe	76	10/85
	Block Out	84	11/85
	Seekrieg per Telefon (Schiffe versenken per Modem)	82	12/85
Spielehilfe	Die Scroll-Maschine — D. Fenster zur Spielewelt (LdM) (+ Fehlert. 11/85)	52	06/85
Sprachen	Tiny Forth Compiler (LdM) (+ Fehlert. 9/85)	51	08/85
Textverarbeitung	Hypra-Text (LdM) (+ Fehlerteufel 11/85)	50	10/85
	Drucksache — Hypra-Text, Teil 2	71	11/85
Tips & Tricks	Große Buchstaben	89	01/85
	Restore für Unterprogramme	90	01/85
Tips & Tricks	Parameterübergabe an Maschinenspracheprogramme	88	01/85
	Cursorsteuerung leicht gemacht	86	02/85
	22 Read Error — Theorie und Praxis	41	03/85
	Floppy-Lister (+ Fehlerteufel 4/85)	82	03/85
	Longscreen beim VC 20	83	05/85
	C 16: Help und Trace verbessert	84	05/85
	Ordnung ist das halbe Leben (Directory-Sorter)	77	05/85
	Dokumentationshilfe, Cross-Referenz-Liste' C 64 (Wettbewerb)	155	06/85
	Prost mit dem C 64: Gerätesteuerung über Userport (+ Fehlerteufel 9/85)	76	06/85
	Fenster-Befehle für den C 16	84	07/85
	Elektronische Merkzettel	83	07/85
	File-Compactor	82	07/85
	REM-Killer (+ Fehlerteufel 9/85)	75	07/85
	Basic-Start-Generator	74	07/85
	Komfortable Ein-/Ausgaberoutine	77	07/85
	Bildschirmmasken leicht erstellt	86	08/85
	Der Bitmap-Compander (HiRes-Bilder komprimieren)	81	08/85
	Hypra-Save	79	08/85
	'Procedure' — oder der C 64 kann lernen	78	08/85
	Aufgewickelt — Listingscrolling für VC 20	63	09/85
	Programmgenerator für den C 64	86	10/85
	Cross-Ref optimiert	83	10/85
	Spieletrainer: Spritekill	86	11/85
	Tipp-Utility	99	12/85
	Der EPROM-Automat (wie man Module macht)	90	12/85
	80-Zeichen-Grafik für den C 128	78	12/85
	Hyper Screen (Sprites auf dem Bildschirmrand)	76	12/85
Transfer	Der C 64 als PET: PET-Simulator	87	01/85
Unterprogramme	Formatierte Eingabe	156	01/85

Software-Tests

Stichwort	Titel	Seite	Ausgabe
Assembler	Assembler im Test Teil 1	34	01/85
Basic-Erweiterung	GBasic — Alles drin	28	01/85
	Macro-Basic: Die Unterprogramm-Bibliothek	137	06/85
	Darf es etwas mehr sein? — Test Business-Basic	120	08/85
	Das Intellectool	138	09/85
	Formel 64: Das Multitalent	158	12/85
DFÜ	Terminalprogramme: Übersicht	42	06/85
Datei	Vergleichstest — 7 Dateiverwaltungen auf einen Blick	118	07/85
	Aufgeräumt mit Mainfile II	157	10/85
Grafik	Malen auf dem Bildschirm (Malprogramme)	34	08/85
	Grafikprogramme auf einen Blick: Marktübersicht	38	08/85
	Vergleichstest: Grafik-Erweiterungen	37	09/85
Lernen	Softlearning — die weiche Welle des Lernens	40	01/85
	Vokabeltraining mit dem Computer	39	03/85
	Marktübersicht: Lernsoftware	168	10/85
Musik	Musik für den C 64: Übersicht Musiksoftware	26	09/85
	The Music System — Zwei auf einen Schlag	164	12/85
Sprachen	Logo — die Sprache für Einsteiger	135	05/85
	Der Ada Trainingskurs auf dem C 64	129	06/85
	Promal — die neue Sprache für Profis?	124	07/85
	Forth-wärts mit M&T-Forth 64	126	07/85
	Was leistet Pilot?	121	08/85
	Pascal für Profis (Profi-Pascal)	122	08/85
	Super-Forth 64	144	09/85
	C — die professionelle Programmiersprache für den C 64	140	09/85
	Basic 7.0 — Das Superbasic des C 128	18	10/85
	Comal 80 — die universelle Programmiersprache	151	10/85
	Turbo-Pascal auf dem C 128	30	11/85

Stichwort	Titel	Seite	Ausgabe
Textverarbeitung	Homeword · Textverarbeitung zu Hause	36	03/85
	Totl-Text — Flexibilität ist Trumpf	38	03/85
	Protext — Textprofi mit 80 Zeichen	133	05/85
	Textomat Plus kontra Vizawrite	132	06/85
	Der Preishammer (Test: StarTexter)	135	09/85
	Paperclip — ausdrücklich gut	44	11/85

So machen's andere

Stichwort	Titel	Seite	Ausgabe
Semmeln	Semmelservice mit dem C 64	147	06/85
Sport	Commodore Sportservice: Heimcomputer zur Turnierauswertung	157	07/85
Hilfe	Computer für Behinderte	182	12/85

Die Ausgaben 2/85 und 4/85 sind bereits vergriffen und nicht mehr lieferbar!

Am besten gleich mitbestellen: Die praktischen 64'er-Sammelboxen



Ein kompletter Jahrgang (12 Ausgaben) paßt in eine der praktischen Sammelboxen! Am besten gleich mitbestellen!

Für alle Leser, die »64'er« regelmäßig kaufen, sammeln oder im Abonnement beziehen, gibt es jetzt ein interessantes Service-Angebot: die 64'er-Sammelbox!

Mit dieser Sammelbox bringen Sie nicht nur Ordnung in Ihre wertvollen Hefte, sondern schaffen sich gleichzeitig ein interessantes und attraktives Nachschlagewerk.

Übrigens: Die Sammelbox ist nicht nur ein praktisches Aufbewahrungsmittel: Sie eignet sich auch hervorragend als Geschenk für Freunde und Bekannte zu vielen Anlässen.

Auch die bisher erschienenen Sonderhefte können Sie jetzt direkt bestellen:

SONDERHEFT 01/84: TIPS & TRICKS
Unentbehrliche Anwendungslistings für C 64 und VC 20.

SONDERHEFT 02/85: ABENTEUERSPIELE 1
Fesselnde Adventures mit zahlreichen Lösungen und einem Programmierkurs.

SONDERHEFT 03/85: SPIELE
Heiße Listings für Spiele-Fans und eine große Marktübersicht.

SONDERHEFT 04/85: GRAFIK & DRUCKER
Von der 3D-Darstellung bis zur Hardcopy-Routine.

SONDERHEFT 05/85: FLOPPY/DATASETTE
Soft-Tools zum komfortablen und noch schnelleren Betrieb von Floppy und Datasette.

SONDERHEFT 06/85: AUSGEWÄHLTE SUPER-LISTINGS
Top-Themen aus 64'er bringt eine Auswahl der besten 64'er Programme.

SONDERHEFT 07/85: ANWENDUNGEN/DFÜ
Leistungsfähige Programme für professionelle Anwendungen und Datenfernübertragung.

SONDERHEFT 08/86: ASSEMBLER
Assembler-Know-how für Anfänger und Fortgeschrittene.

SONDERHEFT 01/86: PC 128
Komplette Beschreibungen von C 128 und C 128D und passendem Zubehör. Die Unterschiede zum C 64.

SONDERHEFT 02/86: TIPS & TRICKS
Super-Listings, ausführliche Grundlagen und die besten Tips&Tricks und Einzeiler aus 64'er.

SONDERHEFT 03/86: C16, C116, VC20 UND PLUS 4
Umfassende Grundlagen und aktuelle Informationen zu C 16, C 116, VC 20 und Plus 4.

SONDERHEFT 04/86: ABENTEUERSPIELE 2
Auf 160 Seiten alles über das Programmieren von Abenteuerspielen und Super-Listings zum Abtippen.

SONDERHEFT 05/86: C64-GRUNDWISSEN
Für alle Einsteiger umfassende Grundlagen und Hilfestellungen rund um den C 64.

SONDERHEFT 06/86: GRAFIK
Grafikprogrammierung des C 64, C 128 und C 128 im C 64-Modus. Dreidimensional konstruieren mit »Giga-CAD«.

Tragen Sie die Nummer des gewünschten Sonderheftes (z.B. 04/85) auf dem Bestellabschnitt der hier eingehefteten Bestell-Zahlkarte ein.

Geschwindigkeitsfreaks können die Routine noch leicht beschleunigen, indem sie den aufwendigen 16-Bit-Vergleich von START(+1) und ENDE(+1) durch einen Zähler ersetzen, der mit der Anzahl der zu summierenden Elemente initialisiert und bei jedem Schleifendurchgang dekrementiert wird. Ein paar weitere Millisekunden pro Schleifendurchgang können eingespart werden, indem JSR KONFAC:RTS durch JMP KONFAC ersetzt wird.

Da es mich reizte, das Programm »Summierung« noch ein wenig auszubauen, stelle ich als »krönenden Abschluß« dieses Artikels das Programm »Statistik« (Listing 12) vor, das die »Varianz« für ein Array innerhalb eines beliebigen Intervalls berechnet.

Für Statistiklaien: Die Varianz ist ein »Streuungsmaß« und gibt Auskunft über die Unterschiedlichkeit der Werte einer Stichprobe. Je höher die Werte, desto weniger einheitlich sind die Stichprobendaten.

$$\text{Varianz} = (\sum_{i=1}^{n} (xi)2) - (\sum_{i=1}^{n} (xi)2)/N/N-1$$

Zwei Summen müssen berechnet werden: S1, die Summe aller quadrierten Werte und S2, die Summe aller Werte. Danach wird S2 durch N (N=Anzahl der Werte) geteilt und das Ergebnis von S1 subtrahiert. Das Resultat wird anschließend durch N-1 geteilt.

Nachfolgend ein Basic-Programm, das ein Array mit 1000 Elementen initialisiert, die Varianz sowohl mit der Assemblerroutine als auch in Basic berechnet und die benötigten Zeiten ausgibt:

```
100 DIM A(1000):LG=1:RG=1000:FOR I=LG TO RG:A(I)=
```

Name	Funktion	Vorbereitung	Ergebnis	C64	C16	C128
EINPOS	POS.1-INT.=>FLIESSK.	ÜBERGABE IN Y	RESULT IM FAC	$B3A2	$9A81	
EINNEG	Neg.1-Int.=>FLIESSK.	ÜBERGABE IM AKKU	RESULT IM FAC	$BC3C	$A2C1	
ZWEPOS	Pos.2-Int.=>FLIESSK.	MANT.1/2:H/L;X:$90;SEC	RESULT IM FAC	$BC49	$A2CE	$8C75
ZWENEG	Neg.2-Int.=>FLIESSK.	MANT.1/2:H/L;X:$90;CLC	RESULT IM FAC	$BC44	$A2C9	$8C70
FLIINT	FLIESSK. NACH INT.	FLIESSKOMMAZAHL IM FAC	RES.IN MANTISSE	$BC9B	$A327	$8CC7
FACSTR	FAC IN STRING	FLIESSKOMMAZAHL IM FAC	STRING+POINTER	$BDDD	$A46F	$8E42
STROUT	STRING AUSGEBEN	NACH FACSTR VORHANDEN	AUSGABE	$AB1E	$9088	$55E2
INTOUT	AUSGABE 2BYTE-INT.	X=LOW-BYTE/A=HIGH-BYTE	AUSGABE	$BDCD	$A45F	$8E32
ADD	FAC=ARG+FAC	FAC-EXPONENT IM AKKU	RESULT IM FAC	$B86A	$9E9E	$8848
SUB	FAC=ARG-FAC	FAC-EXPONENT IM AKKU	RESULT IM FAC	$B853	$9E87	$8831
MULT	FAC=ARG*FAC	FAC-EXPONENT IM AKKU	RESULT IM FAC	$BA2B	$A07B	$8A27
DIV	FAC=ARG/FAC	FAC-EXPONENT IM AKKU	RESULT IM FAC	$BB12	$A197	$8B4C
EXP	FAC=ARGFAC	FAC-EXPONENT IM AKKU	RESULT IM FAC	$BF7B	$A5EE	
FAC*10	FAC=FAC*10	FAC-EXPONENT IM AKKU	RESULT IM FAC	$BAE2	$A162	$8B17
FAC/10	FAC=FAC/10	FAC-EXPONENT IM AKKU	RESULT IM FAC	$BAFE	$A183	$8B38
SIN(X)	SINUS-FUNKTION	ARGUMENT X IM FAC	RESULT IM FAC	$E26B	$AA77	
COS(X)	COSINUS-FUNKTION	ARGUMENT X IM FAC	RESULT IM FAC	$E264	$AA70	
SQR(X)	WURZEL-FUNKTION	ARGUMENT X IM FAC	RESULT IM FAC	$BF71	$A5E4	
LOG(X)	LOG.-FKT. (BASIS E)	ARGUMENT X IM FAC	RESULT IM FAC	$B9EA	$A01E	
EXP(X)	EXPONENTIALFUNKTION	ARGUMENT X IM FAC	RESULT IM FAC	$BFED	$A660	
FACARG	FAC NACH ARG	FLIESSKOMMAZ. IM FAC	FAC NACH ARG	$BC0C	$A291	
ARGFAC	ARG NACH FAC	FLIESSKOMMAZA. IM ARG	ARG NACH FAC	$BBFC		
FACKON	FAC NACH ADRESSE	POINTER (X=L/Y=H)	FAC NACH ADR	$BBD4	$A24C	
KONFAC	KONSTANTE NACH FAC	POINTER (AKKU=L/Y=H)	KONST. IM FAC	$BBA2	$A21F	
KONADD	FAC=KONSTANTE+FAC	POINTER (AKKU=L/Y=H)	RESULT IM FAC	$B867	$9E9B	
KONMUL	FAC=KONSTANTE*FAC	POINTER (AKKU=L/Y=H)	RESULT IM FAC	$BA28	$A078	
CHKKOM	KOMMA LESEN	KEINE	KEINE	$AEFD	$9491	$795C
GETBYT	1BYTE-INTEGER LESEN	KEINE	X=BYTE	$B79E	$9D84	$87F4
FRMNUM	NUM.AUSDRUCK AUSWERT.	KEINE	RESULT IM FAC	$AD8A	$9314	$FFD7
ADRFOR	FLIESSK.NACH 2-INT.	FLIESSKOMMA IM FAC	Y=LOW/AKKU=HIGH	$B7F7	$9DE4	$8815
GETADR	2BYTE-INTEGER LESEN	KEINE	Y=LOW/AKKU=HIGH			$880F
ADRBYT	2BYTE+1BYTE LESEN	KEINE	$14/15 UND X	$B7EB	$9DD2	$8803
GETPOS	POINTER AUF VARIABLE	KEINE	POINTER (AKKU/Y)	$B08B	$96A5	$7AAF

Tabelle: Vergleichstabelle aller bekannten Funktionen und Adressen für C16, C64, C128



```
I:NEXT:PRINT"START!"
110 :
120 TA=TI:SYS 49152,A(LG),A(RG),V
130 PRINT"V="V:TB=TI:PRINT"ASS.:"(TB-TA)/60
140 :
150 TA=TI
160 FOR I=LG TO RG
170 S1=S1+A(I)*A(I):S2=S2+A(I)
180 NEXT
190 V=(S1-S2*S2/(RG-LG+1))/(RG-LG)
200 TB=TI
210 PRINT"BASIC:"(TB-TA)/60
```

Die Assemblerroutine benötigt knapp drei Sekunden, die Basic-Routine knapp 16 Sekunden, um die Varianz für 1000 Elemente zu berechnen.

Mit den Kenntnissen, die Sie nun besitzen, und der nachfolgenden Erläuterung von zwei neuen im Programm verwendeten Routinen sollten Sie in der Lage sein, das Programm zu durchschauen und vielleicht auch um die Berechnung weiterer Kennwerte zu erweitern.

Diese zusätzlich verwendeten Routinen vereinfachen vor allem den Umgang mit Konstanten und erlauben es, den FAC direkt mit einer Konstanten zu multiplizieren oder eine Konstante zum FAC zu addieren. Die Routinen erwarten Konstanten, die – ebenso wie Basic-Variablen – im gepackten Fünf-Byte-Format vorliegen (Vorzeichen in Mantisse2 integriert).

Schauen Sie sich bitte in jedem Fall an, wie »Statistik« die errechnete Varianz an das Basic-Programm übergibt. Mit GETPOS wird ein Zeiger auf die im SYS-Befehl angegebene Variable V geholt und mit der neuen Routine FACKON der Inhalt des FAC in diese Basic-Variable kopiert. Komfortabler kann eine Ergebnisübergabe an Basic wohl kaum sein.

KONMUL ($BA28)

KONMUL multipliziert den FAC mit einer Konstanten, auf die in Akku und Y-Register ein Zeiger übergeben werden muß.

KONADD ($B867)

KONPLU addiert FAC und die Konstante, auf die ein ebenfalls in Akku und Y-Register übergebener Zeiger weist.

(Said Baloui/do)




```
ü
0100            ;*** ZAHLENAUSGABE ***
0110            ;
0120            ;AUSGABE EINER FLIESSKOMMAZAHL
0130            ;AUF DEM BILDSCHIRM.
0140            ;
0150            ;1.FLIESSKOMMAZAHL NACH FAC
0160            ;2.AUFRUF VON FAC=>ASCII
0170            ;3. AUFRUF VON ASCII=>SCREEN
0180            ;
0190FAC         .DE $61
0200KONFAC      .DE $BBA2
0210FACSTR      .DE $BDDD
0220STROUT      .DE $AB1E
0230            ;
0240            .BA $C000             ;PROGRAMMSTART
0250            .OS                   ;CODE GENERIEREN
0260            ;
0270            ;*ZAHL NACH FAC UEBERTRAGEN*
0280            LDA #L,ZAHL
0290            LDY #H,ZAHL
0300            JSR KONFAC
0310            ;
0320            ;*ZAHL AUSGEBEN*
0330            JSR FACSTR
0340            JSR STROUT
0350            RTS
0360            ;
0370            ;*AUSZUGEBENDE ZAHL (PI)*
0380ZAHL        .BY $82 $49 $0F $DA $A1
0390            .EN
//
ü
```

Listing 1. Assemblerlisting für Zahlenausgabe

```
ü
0100            ;*** EIN-BYTE-INTEGER ***
0110            ;
0120            ;KONVERTIERUNG EINER EIN-BYTE-
0130            ;INTEGER-ZAHL INS FLIESSKOMMA-
0140            ;FORMAT UND AUSGABEE AUF DEM
0150            ;BILDSCHIRM
0160            ;
0170            ;1.INTEGERZAHL NACH FLIESSK.
0180            ;2.AUFRUF VON FAC=>ASCII
0190            ;3.AUFRUF VON ASCII=>SCREEN
0200            ;
0210EINPOS      .DE $B3A2
0220EINNEG      .DE $BC3C
0230FACSTR      .DE $BDDD
0240STROUT      .DE $AB1E
0250            ;
0260            .BA $C000             ;PROGRAMMSTART
0270            .OS                   ;CODE GENERIEREN
0280            ;
0290            ;*POSITIVE INT-ZAHL WANDELN*
0300            LDY #$80
0310            JSR EINPOS
0320            ;
0330            ;*ZAHL AUSGEBEN*
0340            JSR FACSTR
0350            JSR STROUT
0360            ;
0370            ;*NEGATIVE INT-ZAHL WANDELN*
0380            LDA #$80
0390            JSR EINNEG
0400            ;
0410            ;*ZAHL AUSGEBEN*
0420            JSR FACSTR
0430            JSR STROUT
0440            RTS
0450            ;
0460            .EN
//
```

Listing 2. Assemblerlisting für Ein-Byte-Integer-Werte

```
ü
0100            ;*** ZWEI-BYTE-INTEGER ***
0110            ;
0120            ;KONVERTIERUNG EINER ZWEI-BYTE-
0130            ;INTEGER-ZAHL INS FLIESSKOMMA-
```

```
0140          ;FORMAT UND AUSGABEE AUF DEM
0150          ;BILDSCHIRM
0160          ;
0170          ;1.INTEGERZAHL NACH FLIESSK.
0180          ;2.AUFRUF VON FAC=>ASCII
0190          ;3.AUFRUF VON ASCII=>SCREEN
0200          ;
0210ZWEIPOS   .DE $BC49
0220ZWEINEG   .DE $BC44
0230FACSTR    .DE $BDDD
0240STROUT    .DE $AB1E
0250          ;
0260          .BA $C000          ;PROGRAMMSTART
0270          .OS                ;CODE GENERIEREN
0280          ;
0290          ;*POSITIVE INT-ZAHL WANDELN*
0300          LDA #$FF           ;ZAHL: $FFFF
0310          STA $62            ;HIGH-BYTE ($FF)
0320          STA $63            ;LOW-BYTE ($FF)
0330          LDX #$90           ;X-REG.: $90
0340          SEC                ;CARRY SETZEN
0350          JSR ZWEIPOS
0360          ;
0370          ;*ZAHL AUSGEBEN*
0380          JSR FACSTR         ;=> AUSGABE VON
0390          JSR STROUT         ;   65535
0400          ;
0410          ;*NEGATIVE INT-ZAHL WANDELN*
0420          LDA #$FF           ;ZAHL: $FFFF
0430          STA $62            ;LOW-BYTE ($FF)
0440          STA $63            ;HIGH-BYTE ($FF)
0450          LDX #$90           ;X-REG.: $90
0460          CLC                ;CARRY LOESCHEN
0470          JSR ZWEINEG
0480          ;
0490          ;*ZAHL AUSGEBEN*
0500          JSR FACSTR         ;=> AUSGABE VON
0510          JSR STROUT         ;   -1
0520          RTS
0530          ;
0540          .EN
//

ü
```

Listing 3. Assemblerlisting für Zwei-Byte-Integer Werte

```
ü
0013          ;
0100          ;*** FLIESSKOMMA ***
0110          ;
0120          ;WANDLUNG EINER FLIESSKOMMAZAHL
0130          ;INS INTEGERFORMAT UND
0140          ;AUSGABE AUF DEM BILDSCHIRM
0150          ;
0160          ;1.FLIESSKOMMAZAHL NACH FAC
0170          ;2.AUFRUF VON FLIESS=>INTEGER
0180          ;3.AUFRUF VON FAC=>STRING
0190          ;4.AUFRUF VON STRING=>SCREEN
0200          ;
0210KONFAC    .DE $BBA2
0220FLIINT    .DE $BC9B
0230EINPOS    .DE $B3A2
0240FACSTR    .DE $BDDD
0250STROUT    .DE $AB1E
0260          ;
0270          .BA $C000          ;PROGRAMMSTART
0280          .OS                ;CODE GENERIEREN
0290          ;
0300          ;*ZAHL NACH FAC UEBERTRAGEN*
0310          LDA #L,ZAHL
0320          LDY #H,ZAHL
0330          JSR KONFAC
0340          ;
0350          ;*ZAHL NACH INTEGER WANDELN*
0360          JSR FLIINT
0370          ;
0380          ;*INTEGER NACH FLIESSK. WANDELN*
0390          LDY $65            ;INT-ERGEBNIS LADEN
0400          JSR EINPOS         ;NACH FLIESSK.WANDELN
0410          ;
0420          ;
0430          ;*ZAHL AUSGEBEN*
0440          JSR FACSTR         ;FAC => STRING
0450          JSR STROUT         ;STRING => SCREEN
0460          RTS
0470          ;
0480          ;*AUSZUGEBENDE ZAHL (PI)*
0490ZAHL      .BY $82 $49 $0F $DA $A1
0500          .EN
//
```

Listing 4. Assemblerlisting für Fließkommaroutinen

```
ü

ü
0013          ;
0100          ;*** INTEGERAUSGABE ***
0110          ;
0120          ;DIREKTE AUSGABE EINER
0130          ;POSITIVEN ZWEI-BYTE-ZAHL
0140          ;AUF DEM BILDSCHIRM
0150          ;
0160INTOUT    .DE $BDCD
0170          ;
0180          .BA $C000          ;PROGRAMMSTART
0190          .OS                ;CODE GENERIEREN
0200          ;
0210          ;*AUSGABE VON $F02A (61482)*
0220          LDX #$2A           ;LOW-BYTE: $2A
0230          LDA #$F0           ;HIGH-BYTE: $F0
0240          JSR INTOUT         ;AUSGABE
0250          RTS
0260          ;
0270          .EN
//
```

Listing 5. Assemblerlisting für Integer-Ausgabe

```
ü

ü
0100          ;*** GRUNDRECHNEN ***
0110          ;
0120          ;
0130ADD       .DE $B86A          ;FAC=ARG+FAC
0140SUB       .DE $B853          ;FAC=ARG-FAC
0150MULT      .DE $BA2B          ;FAC=ARG*FAC
0160DIV       .DE $BB12          ;FAC=ARG/FAC
0170FAC       .DE $61
0180ARG       .DE $69
0190EINPOS    .DE $B3A2
0200ZWEPOS    .DE $BC49
0210FACARG    .DE $BC0C
0220FACSTR    .DE $BDDD
0230STROUT    .DE $AB1E
0240BSOUT     .DE $FFD2
0250          ;
0260          .BA $C000          ;PROGRAMMSTART
0270          .OS                ;CODE GENERIEREN
0280          ;
0290          ;*'/': ANTEIL BERECHNEN*
0300          LDA #L,2000        ;2000,LOW
0310          STA $63            ;NACH $63
0320          LDA #H,2000        ;2000,HIGH
0330          STA $62            ;NACH $62
0340          LDX #$90           ;X INITIALIS.
0350          SEC
0360          JSR ZWEPOS         ;2000 => FLIESSK.
0370          JSR FACARG         ;FAC => ARG
0380          LDY #3             ;ANTEIL: 1/3
0390          JSR EINPOS         ;1/3 => FLIESSK.
0400          ;
0410          JSR DIV            ;FAC=2000/3
0420          JSR AUSGABE        ;ERGEBNIS AUSGEBEN
0430          ;
0440          ;
0450          ;*'+': 2.FLAECHE ADDIEREN*
0460          JSR FACARG         ;ERGEBNIS NACH ARG
0470          LDA #L,5286        ;5286,LOW
0480          STA $63            ;NACH $63
0490          LDA #H,5286        ;5286,HIGH
0500          STA $62            ;NACH $62
0510          LDX #$90           ;X INITIAL.
0520          SEC
0530          JSR ZWEPOS         ;5286=>FLIESSK.
0540          ;
0550          JSR ADD            ;FAC=666,66667+5286
0560          JSR AUSGABE        ;FAC=5952,66667
0570          ;
0580          ;
0590          ;*'-': 52 QM SUBTRAHIEREN*
0600          JSR FACARG         ;FAC => ARG
0610          LDY #52            ;52 NACH
0620          JSR EINPOS         ;FAC
0630          ;
0640          JSR SUB            ;FAC=5952,66667-52
0650          JSR AUSGABE        ;FAC=5900,66667
0660          ;
0670          ;
0680          ;*'*': DM-WERT BERECHNEN*
0690          JSR FACARG         ;FAC => ARG
0700          LDY #228           ;228 NACH
0710          JSR EINPOS         ;FAC
```

```
0720                ;
0730                JSR MULT              ;FAC=5900,66667*228
0740                JSR AUSGABE           ;FAC=1345352
0750                RTS
0760                ;
0770                ;
0780                ;*ZAHL AUSGEBEN*
0790AUSGABE         JSR FACPUF            ;FAC RETTEN
0800                JSR FACSTR            ;FAC=>STRING
0810                JSR STROUT            ;STRING=>SCREEN
0820                LDA #13               ;ZEILENVORSCHUB
0830                JSR BSOUT             ;AUSGEBEN
0840                JSR PUFFAC            ;FAC HOLEN
0850                RTS
0860                ;
0870                ;*FAC RETTEN*
0880FACPUF          LDX #5
0890LOOP1           LDA FAC,X
0900                STA PUFFER,X
0910                DEX
0920                BPL LOOP1
0930                RTS
0940                ;
0950                ;*FAC HOLEN*
0960PUFFAC          LDX #5
0970LOOP2           LDA PUFFER,X
0980                STA FAC,X
0990                DEX
1000                BPL LOOP2
1010                RTS
1020                ;
1030PUFFER
1040                .EN
//
```

Listing 6. Assemblerlisting für Grundrechnungsarten

```
0100                ;*** FUNKTIONEN ***
0110                ;
0120                ;
0130COS             .DE $E264
0140SIN             .DE $E26B
0150SQR             .DE $BF71
0160LOG             .DE $B9EA
0170EXP             .DE $BFED
0180FAC             .DE $61
0190ARG             .DE $69
0200EINPOS          .DE $B3A2
0210ZWEPOS          .DE $BC49
0220FACARG          .DE $BC0C
0230FACSTR          .DE $BDDD
0240STROUT          .DE $AB1E
0250BSOUT           .DE $FFD2
0260                ;
0270                .BA $C000             ;PROGRAMMSTART
0280                .OS                   ;CODE GENERIEREN
0290                ;
0300                ;* SIN, COS, SQR, LOG, EXP *
0310                LDY #10
0320                JSR EINPOS
0330                JSR SIN
0340                JSR AUSGABE
0350                ;
0360                LDY #10
0370                JSR EINPOS
0380                JSR COS
0390                JSR AUSGABE
0400                ;
0410                LDY #10
0420                JSR EINPOS
0430                JSR SQR
0440                JSR AUSGABE
0450                ;
0460                LDY #10
0470                JSR EINPOS
0480                JSR LOG
0490                JSR AUSGABE
0500                ;
0510                LDY #10
0520                JSR EINPOS
0530                JSR EXP
0540                JSR AUSGABE
0550                RTS
0560                ;
0570                ;
0580                ;*ZAHL AUSGEBEN*
0590AUSGABE         JSR FACSTR            ;FAC=>STRING
0600                JSR STROUT            ;STRING=>SCREEN
0610                LDA #13               ;ZEILENVORSCHUB
0620                JSR BSOUT             ;AUSGEBEN
0630                RTS
0640                ;
0660                .EN
//
```

Listing 7. Assemblerlisting für Funktionen

```
0100                ;*** USR-VEKTOR ***
0110                ;
0120                ;
0130SQR             .DE $BF71
0150MULT            .DE $BA2B             ;FAC=ARG*FAC
0170FAC             .DE $61
0180ARG             .DE $69
0210FACARG          .DE $BC0C
0250                ;
0260                .BA $C000             ;PROGRAMMSTART
0270                .OS                   ;CODE GENERIEREN
0280                ;
0290                ;*WURZEL(X) BERECHNEN*
0300                JSR SQR               ;WURZEL(X) BERECHNEN
0310                JSR FACPUF            ;ERGEBNIS RETTEN
0320                RTS
0330                ;
0340                ;
0350                .BA $C100             ;START TEIL 2
0360                ;
0370                ;*ZAHL(Y) DURCH WURZEL(X) DIVID.*
0380                JSR FACARG            ;ZAHL(Y) NACH FAC
0390                JSR PUFFAC            ;WURZEL(X) NACH ARG
0400                JSR MULT              ;FAC=ARG*FAC
0410                RTS
0860                ;
0870                ;*FAC RETTEN*
0880FACPUF          LDX #5
0890LOOP1           LDA FAC,X
0900                STA PUFFER,X
0910                DEX
0920                BPL LOOP1
0930                RTS
0940                ;
0950                ;*FAC HOLEN*
0960PUFFAC          LDX #5
0970LOOP2           LDA PUFFER,X
0980                STA FAC,X
0990                DEX
1000                BPL LOOP2
1010                RTS
1020                ;
1030PUFFER
1040                .EN
//
```

Listing 8. Assemblerlisting für USR-Vektor-Funktionen

```
0100                ;*** USR-VEKTOR.2 ***
0110                ;
0120                ;
0130ADD             .DE $B86A
0140FAC             .DE $61
0150ARG             .DE $69
0160FACARG          .DE $BC0C
0170                ;
0180                .BA $C000             ;PROGRAMMSTART
0190                .OS                   ;CODE GENERIEREN
0200                ;
0210                ;*PARAMETER UEBERNEHMEN*
0220                DEC 250
0230                BEQ LOOP3
0240                JSR FACPUF
0250                LDA KORR+1
0260                CLC
0270                ADC #5
0280                STA KORR+1
0290                BCC ENDE
0300                INC KORR+2
0310ENDE            RTS
0320                ;
0330                ;*RECHNEN*
0340LOOP3           JSR FACARG
0350                JSR PUFFAC
0360                JSR ADD
0370                LDA LOOP2+1
0380                CLC
0390                ADC #5
```

```
0400           STA LOOP2+1
0410           BCC SPRUNG
0420           INC LOOP2+2
0430SPRUNG     LDA LOOP2+2
0440           CMP KORR+2
0450           BNE LOOP3
0460           LDA.LOOP2+1
0470           CMP KORR+1
0480           BNE LOOP3
0490           RTS
0500           ;
0510           ;*FAC RETTEN*
0520FACPUF     LDX #5
0530LOOP1      LDA FAC,X
0540KORR       STA PUFFER,X
0550           DEX
0560           BPL LOOP1
0570           RTS
0580           ;
0590           ;*FAC HOLEN*
0600PUFFAC     LDX #5
0610LOOP2      LDA PUFFER,X
0620           STA FAC,X
0630           DEX
0640           BPL LOOP2
0650           RTS
0660           ;
0670PUFFER
0680           .EN
//
```

Listing 9. Assemblerlisting für USR-Vektor2-Funktionen

```
ü

ü
0100           ;*** PARAMETER LESEN ***
0110           ;
0120           ;
0130CHKKOM     .DE $AEFD
0140GETBYT     .DE $B79E
0150FRMNUM     .DE $AD8A
0160ADRFOR     .DE $B7F7
0170ADRBYT     .DE $B7EB
0180FACSTR     .DE $BDDD
0190STROUT     .DE $AB1E
0200INTOUT     .DE $BDCD
0210BSOUT      .DE $FFD2
0220           ;
0230           .BA $C000                ;PROGRAMMSTART
0240           .OS                      ;CODE GENERIEREN
0250           ;
0260           ;*EIN-BYTE-WERT*
0270           JSR CHKKOM
0280           JSR GETBYT               ;EIN-BYTE-WERT
0290           LDA #0                   ;HIGH-BYTE=0
0300           JSR INTOUT               ;AUSGEBEN
0310           LDA #13
0320           JSR BSOUT
0330           ;
0340           ;*ZWEI-BYTE-WERT*
0350           JSR CHKKOM               ;ZWEI-BYTE-WERT
0360           JSR FRMNUM               ;NACH FAC
0370           JSR ADRFOR               ;INS INTEGERFORMAT
0380           PHA                      ;Y=LOW-BYTE
0390           TYA                      ;INS
0400           TAX                      ;X-REGISTER
0410           PLA                      ;UEBERTRAGEN
0420           JSR INTOUT               ;AUSGEBEN
0430           LDA #13
0440           JSR BSOUT
0450           ;
0460           ;*ZWEI- UND EIN-BYTE-WERT*
0470           JSR CHKKOM               ;ZWEI-BYTE-WERT
0480           JSR ADRBYT               ;+EIN-BYTE-WERT
0490           LDA #0                   ;HIGH-BYTE=0
0500           JSR INTOUT               ;1BYTE AUSGEBEN
0510           LDA #13
0520           JSR BSOUT
0530           LDX $14                  ;X=LOW
0540           LDA $15                  ;A=HIGH
0550           JSR INTOUT               ;AUSGEBEN
0560           LDA #13
0570           JSR BSOUT
0580           ;
0590           ;*BELIEBIGER WERT*
0600           JSR CHKKOM               ;BELIEBIGEN NUM.
0610           JSR FRMNUM               ;AUSDRUCK NACH FAC
0620           JSR FACSTR               ;FAC => STRING
0630           JSR STROUT               ;STRING AUSGEBEN
0640           RTS
0650           ;
0660           .EN
//
```

Listing 10. Assemblerlisting für Parameterlesen

```
ü

ü
0100           ;*** SUMMIERUNG ***
0110           ;
0120           ;
0130GETPOS     .DE $B08B                ;=> POINTER
0140CHKKOM     .DE $AEFD
0150START      .DE 251                  ;LINKE GRENZE
0160ENDE       .DE 253                  ;RECHTE GRENZE
0170ADD        .DE $B86A                ;FAC=ARG+FAC
0180SUB        .DE $B853                ;FAC=ARG-FAC
0190MULT       .DE $BA2B                ;FAC=ARG*FAC
0200DIV        .DE $BB12                ;FAC=ARG/FAC
0210FAC        .DE $61
0220ARG        .DE $69
0230KONFAC     .DE $BBA2
0240FACARG     .DE $BC0C
0250FACSTR     .DE $BDDD
0260STROUT     .DE $AB1E
0270BSOUT      .DE $FFD2
0280           ;
0290           .BA $C000                ;PROGRAMMSTART
0300           .OS                      ;CODE GENERIEREN
0310           ;
0320           ;*POINTER HOLEN*
0330           JSR CHKKOM               ;POINTER AUF
0340           JSR GETPOS               ;LINKE GRENZE
0350           STA START                ;NACH START(+1)
0360           STY START+1
0370           JSR CHKKOM               ;POINTER AUF
0380           JSR GETPOS               ;RECHTE GRENZE
0390           STA ENDE                 ;NACH ENDE(+1)
0400           STY ENDE+1
0410           ;
0420           JSR INFAC                ;A(1) NACH FAC
0430           ;
0440BEGIN      JSR FACARG               ;FAC NACH ARG
0450           LDA START
0460           CLC                      ;START(+1)=
0470           ADC #5                   ;START(+1) + 5
0480           STA START                ;<=>POINTER
0490           BCC LABEL1               ;AUF NAECHSTE
0500           INC START+1              ;REALZAHL
0510           ;
0520LABEL1     LDA ENDE+1
0530           CMP START+1
0540           BCC BACK                 ;FERTIG, WENN
0550           BNE WEITER               ;ENDE(+1)<START(+1)
0560           LDA ENDE
0570           CMP START
0580           BCC BACK
0590           ;
0600WEITER     JSR INFAC                ;A(START) => FAC
0610           JSR ADD                  ;FAC=ARG+FAC
0620           JMP BEGIN                ;SCHLEIFENANFANG
0630           ;
0640BACK       JSR FACSTR               ;AUSGABE VON FAC
0650           JSR STROUT               ;(=ERGEBNIS)
0660           RTS
0670           ;
0680INFAC      LDA START                ;KONSTANTE, AUF
0690           LDY START+1              ;DIE START(+1)
0700           JSR KONFAC               ;WEIST, NACH
0710           RTS                      ;ARG KOPIEREN
0720           ;
0730           .EN
```

Listing 11. Assemblerlisting für Sumierung

```
ü

ü
0100           ;*** STATISTIK ***
0110           ;
0120           ;
0130GETPOS     .DE $B08B                ;=> POINTER
0140CHKKOM     .DE $AEFD
0150SUMME1     .DE $A5
0160SUMME2     .DE $AA
```

Listing 12. Assemblerlisting für Statistik (Fortsetzung)

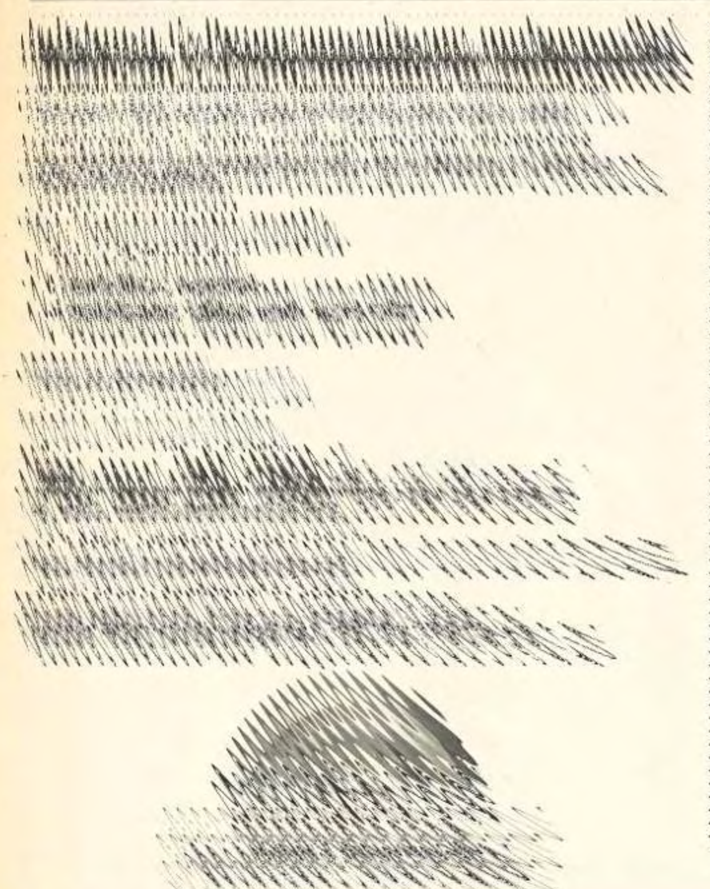
```
0170PUFFER      .DE $B2          ;PUFFER
0180START       .DE $9B          ;LINKE GRENZE
0190STKOP       .DE $9E          ;RECHTE GRENZE
0200COUNT       .DE $B7          ;ZAEHLER
0210N           .DE $B9          ;ZAEHLER
0220N1          .DE $FA          ;ZAEHLER
0230ADD         .DE $B86A        ;FAC=ARG+FAC
0240SUB         .DE $B853        ;FAC=ARG-FAC
0250MULT        .DE $BA2B        ;FAC=ARG*FAC
0260DIV         .DE $BB12        ;FAC=ARG/FAC
0270EXP         .DE $BF7B        ;FAC=ARGßFAC
0280FAC         .DE $61
0290ARG         .DE $69
0300KONFAC      .DE $BBA2
0310KONARG      .DE $BA8C
0320FACKON      .DE $BBD4
0330FACARG      .DE $BC0C
0340KONMUL      .DE $BA28
0350KONADD      .DE $B867
0360FLIINT      .DE $BC9B
0370ZWEPOS      .DE $BC49
0380EINPOS      .DE $B3A2
0390FACSTR      .DE $BDDD
0400STROUT      .DE $AB1E
0410            ;
0420            .BA $C000        ;PROGRAMMSTART
0430            .OS              ;CODE GENERIEREN
0440            ;
0450            ;***POINTER HOLEN***
0460            JSR CHKKOM       ;POINTER AUF
0470            JSR GETPOS       ;LINKE GRENZE
0480            STA START        ;NACH START(+1)
0490            STY START+1
0500            STA STKOP
0510            STY STKOP+1
0520            JSR CHKKOM       ;POINTER AUF
0530            JSR GETPOS       ;RECHTE GRENZE
0540            ;
0550            ;
0560            ;***N, N-1, COUNT ERMITTELN***
0570            SEC
0580            SBC START        ;DIFFERENZ
0590            STA $63          ;ZWISCHEN DEN
0600            TYA              ;GRENZEN BERECHNEN
0610            SBC START+1
0620            STA $62
0630            LDX #$90
0640            SEC
0650            JSR ZWEPOS
0660            JSR FACARG
0670            ;
0680            LDY #5           ;DIFFERENZ GETEILT
0690            JSR EINPOS       ;DURCH 5 ERGIBT
0700            JSR DIV          ;DIE ANZAHL N-1
0710            LDX #L,N1
0720            LDY #H,N1
0730            JSR FACKON
0740            ;
0750            JSR FLIINT       ;N-1 INKREMENTIEREN
0760            LDX $64          ;ERGIBT N
0770            LDY $65
0780            INY
0790            BCC NOINC
0800            INX
0810NOINC       STX N+1
0820            STY N
0830            STX COUNT+1      ;UND GLEICHZEITIG
0840            STY COUNT        ;COUNT (COUNT=N)
0850            ;
0860            LDY #0
0870            JSR EINPOS       ;PUFFER INI-
0880            JSR FACPUF       ;TIALISIEREN
0890            ;
0900            ;
0910            ;***SUMME1 BERECHNEN***
0920LOOPS1      JSR INFAC        ;A(START) NACH
0930            LDA START        ;FAC KOPIEREN
0940            LDY START+1      ;UND FAC MIT
0950            JSR KONMUL       ;A(START) MULTIPL
0960            LDA #L,PUFFER    ;FAC UND SUMME1
0970            LDY #H,PUFFER    ;ADDIEREN UND
0980            JSR KONADD       ;NEUE SUMME1 IN
0990            JSR FACPUF       ;PUFFER KOPIEREN
1000            JSR OFFSET       ;START INKREM.
1010            JSR DECCNT       ;ZAEHLER DEKREM.
1020            BNE LOOPS1       ;FERTIG? NEIN=>
1030            ;
1040            LDA STKOP        ;START WIEDER
1050            LDX STKOP+1      ;AUF DAS ERSTE
1060            STA START        ;ELEMENT A(LG)
1070            STX START+1      ;SETZEN
1080            LDA N
1090            LDX N+1          ;COUNT MIT DER
1100            STA COUNT        ;ANZAHL N LADEN
1110            STX COUNT+1
1120            LDY #0           ;UND FAC
1130            JSR EINPOS       ;INITIALSIEREN
1140            ;
1150            ;
1160            ;*** SUMME2 BERECHNEN ***
1170LOOPS2      LDA START        ;ZU FAC DAS ELE-
1180            LDY START+1      ;MENT A(START)
1190            JSR KONADD       ;ADDIEREN
1200            JSR OFFSET       ;START INKREM.
1210            JSR DECCNT       ;ZAEHLER DEKREM.
1220            BNE LOOPS2       ;FERTIG? NEIN=>
1230            ;
1240            ;
1250            ;*** SUMME2=SUMME2*SUMME2 ***
1260            JSR FACARG       ;SUMME2 NACH ARG
1270            LDY #2           ;ZAHL 2 NACH FAC
1280            JSR EINPOS
1290            JSR EXP          ;FAC=FACßARG
1300            ;
1310            ;
1320            ;*** SUMME2 DURCH N DIVID. ***
1330            JSR FACARG       ;SUMME2 NACH ARG
1340            LDA N            ;BRINGEN
1350            LDX N+1
1360            STA $63          ;N NACH FAC
1370            STX $62          ;BRINGEN
1380            LDX #$90
1390            SEC
1400            JSR ZWEPOS       ;SUMME2 DURCH
1410            JSR DIV          ;N DIVIDIEREN
1420            ;
1430            ;
1440            ;*** SUMME2 VON SUMME1 SUB. ***
1450            LDA #L,PUFFER    ;SUMME1 NACH ARG
1460            LDY #H,PUFFER    ;BRINGEN UND
1470            JSR KONARG       ;SUMME2 VON
1480            JSR SUB          ;SUMME1 SUBTRAH.
1490            ;
1500            ;
1510            ;*** RESULTAT DURCH N-1 DIVID.***
1520            JSR FACARG       ;RESULTAT NACH
1530            LDA #L,N1        ;ARG BRINGEN
1540            LDY #H,N1        ;KONSTANTE N NACH
1550            JSR KONFAC       ;FAC UND ARG
1560            JSR DIV          ;DURCH FAC DIV.
1570            ;
1580            ;
1590            ;*** VARIANZ IN V SPEICHERN ***
1600            JSR CHKKOM       ;ZEIGER AUF V
1610            JSR GETPOS       ;HOLEN UND
1620            TAX              ;DIE BERECHNETE
1630            JMP FACKON       ;VARIANZ NACH V
1640            ;
1650            ;
1660            ;* ZAEHLER DEKREMENTIEREN *
1670DECCNT      LDA COUNT
1680            BNE LAB1
1690            DEC COUNT+1
1700LAB1        DEC COUNT
1710            LDA COUNT+1
1720            BNE DECEND
1730            LDA COUNT
1740DECEND      RTS
1750            ;
1760            ;
1770            ;* A(START) NACH FAC *
1780INFAC       LDA START        ;KONSTANTE, AUF
1790            LDY START+1      ;DIE START(+1)
1800            JMP KONFAC       ;WEIST, NACH
1810            ;
1820            ;* KONST.IN PUFFER NACH FAC *
1830PUFFAC      LDA #L,PUFFER
1840            LDY #H,PUFFER
1850            JMP KONFAC
1860            ;
1870            ;* FAC NACH PUFFER KOPIEREN *
1880FACPUF      LDX #L,PUFFER
1890            LDY #H,PUFFER
1900            JMP FACKON
1910            ;
1920            ;* START AUF NEXT ELEMENT *
1930OFFSET      LDA START
1940            CLC
1950            ADC #5
1960            STA START
1970            BCC OFFEND
1980            INC START+1
1990OFFEND      RTS
2000            ;
2010            .EN
//
```

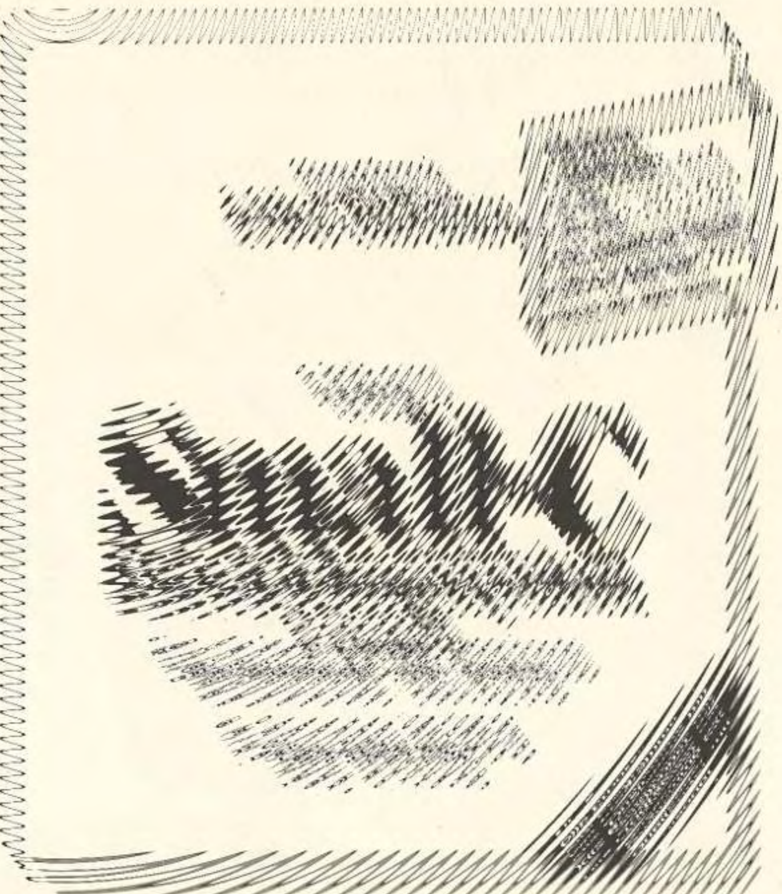
Listing 12. Assemblerlisting für Statistik (Schluß)

Fortsetzung von S. 65

MEMORY MAP mit Wandervorschlägen

TAN		
18	$12	Flagge für Vorzeichen des Ergebnisses bei SIN und TAN
Tastatur		
145	$91	Zwischenspeicher für Abfrage der STOP-Taste
197	$C5	Tasten-Code der zuletzt gedrückten Taste
198	$C6	Anzahl der Zeichen im Tastaturpuffer
203	$CB	Tasten-Code der gerade gedrückten Taste
208	$D0	Flagge für Eingabe von Tastatur oder Bildschirm
245-246	$F5-$F6	Vektor auf die Decodiertabelle für ASCII-Codewerte der Tasten
631-640	$277-$280	Tastaturpuffer
649	$289	Maximale Länge des Tastaturpuffers
650	$28A	Flagge für Tastenwiederholung
651	$28B	Zähler für Wiederholungsgeschwindigkeit der Tasten
652	$28C	Zähler für Ansprechzeit der Wiederholfunktion von Tasten
653	$28D	Tastencode der SHIFT-, CTRL- und Commodore-Taste
654	$28E	Tastencode der zuletzt gedrückten SHIFT-, CTRL- und Commodore-Taste
655-656	$28F-$290	Vektor auf die Routine der Tastencode-Tabellen
657	$291	Flagge für Verriegelung der Zeichensatz-Umschaltung
Token		
8	$8	Suchzeichen speziell für Befehlsende und Gänsefüße
11	$B	Anzahl der Dimensionen von Feldern (Arrays)
15	$F	Flagge bei LIST, Garbage Collection und Textumwandlung
61-62	$3D-$3E	Zeiger auf die Adresse, ab welcher der Text der laufenden Basic-Zeile abgespeichert ist
122-123	$7A-$7B	Teile der CHRGET-Routine
512-600	$200-$258	Basic-Eingabepuffer
772-773	$304-$305	Indirekter Sprungvektor auf die Basic-Routine, die ASCII-Text in Tokens umwandelt
774-775	$306-$307	Indirekter Sprungvektor auf die Basic-Routine, die Tokens in ASCII-Text zurückwandelt (List)
776-777	$308-$309	Indirekter Sprungvektor auf die Basic-Routine, die den nächsten Befehl liest und ausführt
Uhr		
160-162	$A0-$A2	Interne Uhr für TI und TI$
USR		
784-786	$310-$312	Nur C64; Sprungbefehl und wählbare Sprungadresse des USR-Befehls
0-2	$0-$2	Nur VC 20: Sprungbefehl und wählbare Sprungadresse des USR-Befehls
Variable		
13	$D	Flagge zur Bestimmung des Variablentyps (String oder Zahl)
14	$E	Flagge zur Bestimmung des Variablentyps (ganze Zahl oder Gleitkommazahl)
45-46	$2D-$2E	Zeiger auf die Anfangsadresse des Speicherbereichs für Variable
47-48	$2F-$30	Zeiger auf die Anfangsadresse des Speicherbereichs für Felder (Arrays)
49-50	$31-$32	Zeiger auf die Endadresse +1 des Speicherbereichs für Felder (Arrays)
51-52	$33-$34	Zeiger auf die untere Grenze des Speicherbereichs für den Text der Stringvariablen
53-54	$35-$36	Zeiger auf die Adresse des zuletzt eingegebenen Strings
69-70	$45-$46	Name der gerade aufgerufenen Basic-Variablen
71-72	$47-$48	Zeiger auf die Adresse des Wertes der gerade aufgerufenen Basic-Variablen
73-74	$49-$4A	Zwischenspeicher für Variable einer FOR-NEXT-Schleife für diverse Basic-Befehle
Vektoren für indirekte Sprünge		
768-779	$300-$30B	Vektoren auf Routinen des Basic-Übersetzers (Interpreters)
794-819	$31A-$333	Vektoren auf Routinen des Betriebssystems (Kernel)





84 KByte Speicher sicher im Griff

Durch einfaches Verändern eines Registers können beim C64 normalerweise brachliegende Speicherbereiche genutzt werden. Um jedoch einen Systemabsturz auszuschließen, ist es notwendig, die auf diese Weise erzeugten Speicherkonfigurationen zu kennen.

Der C64 besitzt bekanntlich durchgehend 64 KByte RAM (Schreib-Lese-Speicher = Random Access Memory) und 20 KByte ROM (Nur-Lese-Speicher = Read Only Memory). Jeweils 8 KByte ROM entfallen auf den Basic-Interpreter ($A000-$BFFF) und das Kernel-ROM ($E000-$FFFF). Weitere 4 KByte ROM sind für die Ein-/Ausgabesteuerung des Computers ($D000-$DFFF) zuständig. Derselbe Speicherbereich enthält auch den Zeichensatz. Da der Prozessor des C64 jedoch maximal 64 KByte Speicher verwalten kann, ist es erforderlich, den darüber hinausgehenden Speicher auszublenden. Der Inhalt der Speicherstelle 1 in der Zeropage zeigt an, welche Speicherbereiche gerade aktiv sind (ROM oder RAM). In diesem Zusammenhang sind lediglich die drei unteren Bits von Bedeutung:
Bit 0 LORAM (Basic-ROM oder RAM)
Bit 1 HIRAM (Kernel-ROM oder RAM)
Bit 2 CHAREN (I/O = Ein-/Ausgabe, Zeichensatz oder RAM)
(Die Bits 3 bis 5 steuern den Kassettenport, Bit 6 und Bit 7 sind nicht benutzt.)

Hieraus ergeben sich acht verschiedene Bit-Kombinationen und somit acht unterschiedliche Speicherkonfigurationen. Im Einschaltzustand enthält die Speicherstelle 1 den Wert 55 (hexadezimal $37, binär 00110111), das heißt, die drei ausschlaggebenden Bits sind gesetzt. In diesem Fall sind Basic-Interpreter, Betriebssystem und Ein-/Ausgabevorrichtung eingeblendet, das RAM unter dem ROM und der Zeichensatz nicht auslesbar (Bild 1).

Wird Bit 0 mit »POKE 1,54« gelöscht, ist das Basic-ROM abgeschaltet und das darunter liegende RAM lesbar (Bild 2). Dieser Befehl ist deshalb nur dann von Basic aus sinnvoll, wenn das Basic-ROM zuvor ins RAM kopiert wurde. Andernfalls stürzt der Computer ab.

Der Wert 53 (hexadezimal $35) bewirkt, daß neben dem Basic-ROM auch das Kernel abgeschaltet und gegen das RAM ausgetauscht ist (Bild 3). In diesem Fall stehen dem Anwender auch keine Betriebssystem-Routinen mehr zur Verfügung. Es sei denn, das Kernel wurde zuvor ins RAM kopiert oder ein modifiziertes Betriebssystem geladen.

Enthält das Register den Wert 52 (hexadezimal $34), wird auch der Ein-/Ausgabe-Baustein ausgeblendet (Bild 4): Der gesamte Speicher des C64 besteht nun ausschließlich aus RAM (64 KByte).

Bei den Werten 51 bis 48 (Bild 5 bis 8) ist – im Gegensatz zu den bisher genannten – Bit 2 (CHAREN) gelöscht. Dies hat zur Folge, daß anstelle des Ein-/Ausgabe-Bausteins der Zeichensatz auslesbar ist.

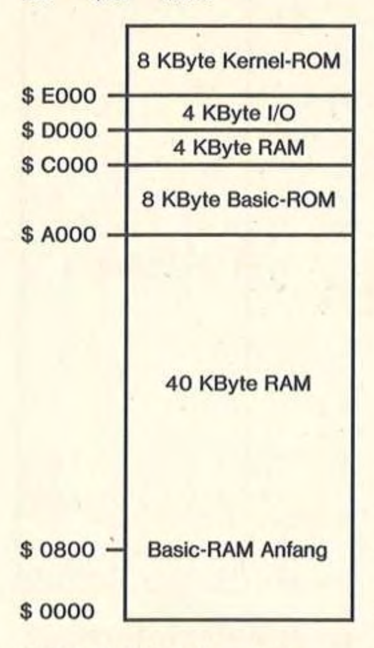


Bild 1. C64 im Einschaltzustand (16 KByte ROM, 44 KByte RAM, 4 KByte I/O)

Inhalt der Speicherstelle 1
(Prozessorport)
DEC HEX BIN untere drei Bits
54 $36 %110
8 KByte Kernel-ROM
$ E000
4 KByte I/O
$ D000
$ C000
$ A000
52 KByte RAM
$ 0800
$ 0000

Bild 2. Basic-ROM ausgeblendet (8 KByte Kernel, 52 KByte RAM, 4 KByte I/O)

Inhalt der Speicherstelle 1
(Prozessorport)
DEC HEX BIN untere drei Bits
53 $35 %101
8 KByte RAM
$ E000
4 KByte I/O
$ D000
$ C000
$ A000
52 KByte RAM
$ 0800
$ 0000

Bild 3. Basic-ROM und Kernel abgeschaltet (60 KByte RAM, 4 KByte I/O)

Inhalt der Speicherstelle 1
(Prozessorport)
DEC HEX BIN untere drei Bits
52 $34 %100
$ E000
$ D000
$ C000
$ A000
64 KByte RAM
$ 0800
$ 0000

Bild 4. Der Speicher besteht aus 64 KByte RAM. Der Computer ist stillgelegt.

Besonders erwähnenswert ist in diesem Zusammenhang der Register-Inhalt 51 (hexadezimal $33). Er ermöglicht (von Basic aus!), den Commodore-Zeichensatz ins RAM zu kopieren (Listing 1). Basic-Interpreter und Betriebssystem bleiben aktiviert (Bild 5). Allerdings ist es vor dem Befehl »POKE 1,51« unbedingt erforderlich, den Interrupt zum Beispiel mit »POKE 56333,127« abzuschalten und nach dem Zurücksetzen (mit »POKE 1,55«) durch »POKE 56333,129« wieder zuzulassen.

```
10 POKE 56333,127 : REM INTERRUPT VERHINDE
   RN                                      <059>
20 POKE 1,51 : REM BIT 2 LOESCHEN          <207>
30 FOR I=0 TO 4095                         <088>
50 POKE 3*4096+I,PEEK(53248+I)             <209>
60 NEXT                                    <070>
70 POKE 1,55 : REM BIT 2 SETZEN            <254>
80 POKE 56333,129 : REM INTERRUPT ZULASSEN <007>
90 POKE 53272,28 : REM ZEICHENSATZ IM RAM
   AKTIVIEREN                              <004>
```

Listing 1. Basic-Programm zum Kopieren des Zeichensatzes ins RAM

Unbegrenzte Möglichkeiten

Vor dem Ausblenden bestimmter Speicherbereiche sollte man sich versichern, daß die im Programm nachfolgend verwendeten Kommandos auch vom Computer ausgeführt werden können. Das gilt insbesondere für Kernel-ROM, Basic Interpreter und den Ein-/Ausgabebereich. So ist beispielsweise das Ändern der Bildschirmfarbe oder die Ausgabe von Tönen bei abgeschaltetem Ein-/Ausgabe-Baustein unmöglich. Bei einer Speicherkonfiguration, die ausschließlich aus RAM besteht, können lediglich Maschinensprache-Kommandos verwendet werden! Diese Speicheranordnung wird zeitweilig von professionellen Textverarbeitungsprogrammen verwendet, die den vorhandenen Speicher optimal nutzen. Basic-Erweiterungen (zum Beispiel »Simons Basic«) verwenden häufig das RAM unter dem Basic-ROM für zusätzliche Kommandos.

Programme, die mit modifiziertem Basic-Interpreter und/oder Betriebssystem arbeiten (beispielsweise auch der 64'er Checksummer) benötigen den Register-Inhalt 53.

Ein weiterer, häufig anzutreffender Einsatzbereich ist die Speicherung von Zeichensätzen und hochauflösenden Grafiken in dem RAM-Bereich ab Adresse $E000. Das leistungsfähige Grafikprogramm »Hi Eddi« (64'er, Ausgabe 1/85) arbeitet gar mit bis zu sieben Grafikbildschirmen, wobei der gesamte RAM-Bereich ab Adresse $2000 genutzt wird.

Bemerkenswert ist schließlich noch, daß sich das Kernel-ROM nicht ausblenden läßt, ohne gleichzeitig das Basic-ROM abzuschalten. Will man deshalb von Basic mit verändertem Betriebssystem arbeiten – und keinen Eingriff in die Hardware des C 64 vornehmen –, muß der Basic-Interpreter und das Kernel-ROM unbedingt zuvor als Kopie im RAM stehen. Listing 2 kopiert diese beiden Speicherbereiche in weniger als einer Sekunde. (nj)

```
10 DATA 169,  0,162,160.133, 95,134, 96,16
   9                                       <125>
15 DATA 0,162,192,133, 90,134, 91,169,  0  <088>
20 DATA 162.192.133, 88.134. 89. 76.191,16
   3                                       <196>
30 FOR I=0 TO 26:READ A:POKE 828+I,A:NEXT:
   SYS(828)                                <161>
40 POKE 831,224:POKE 839,0:POKE 847,0:SYS(
   828)                                    <170>
```

Listing 2. Basic-Lader zum Kopieren von Basic- und Kernel-ROM ins RAM

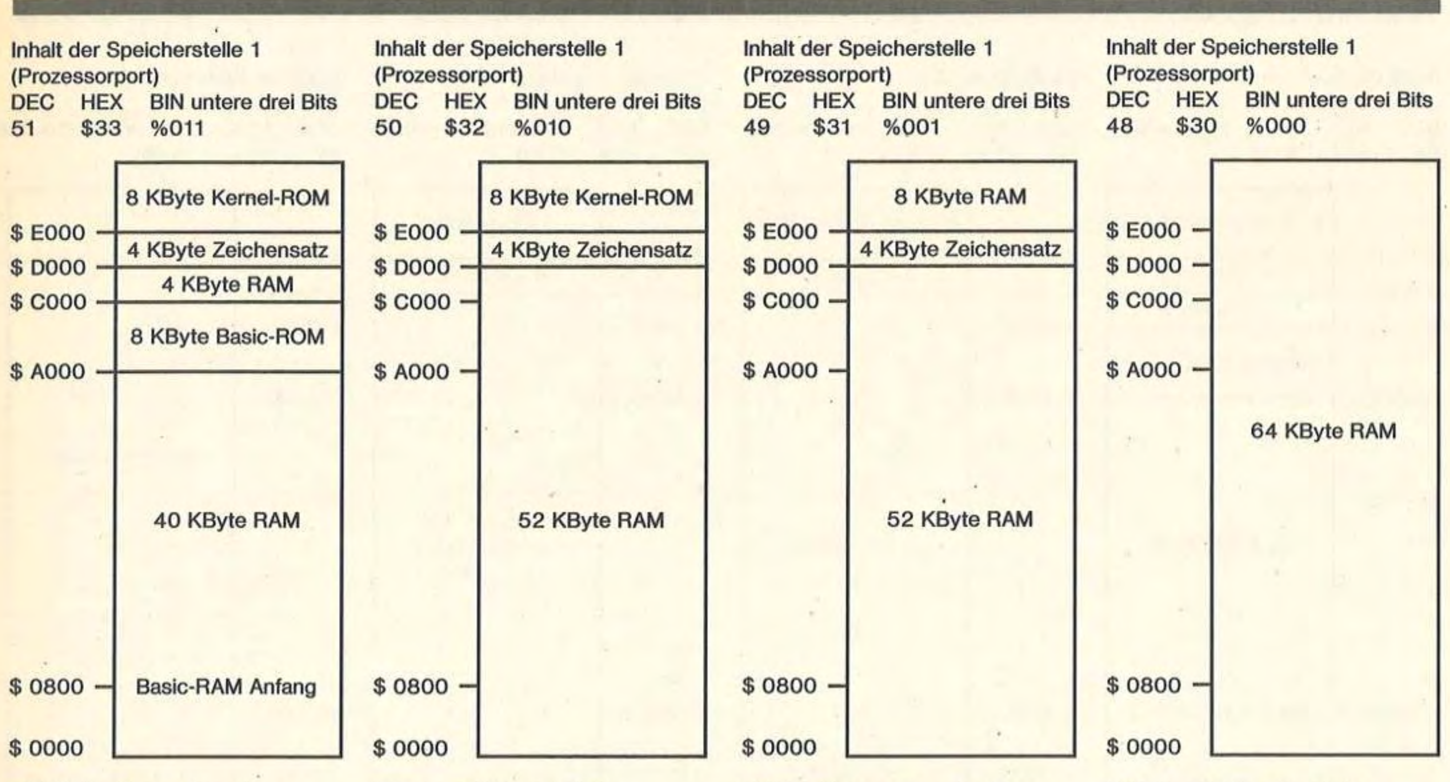
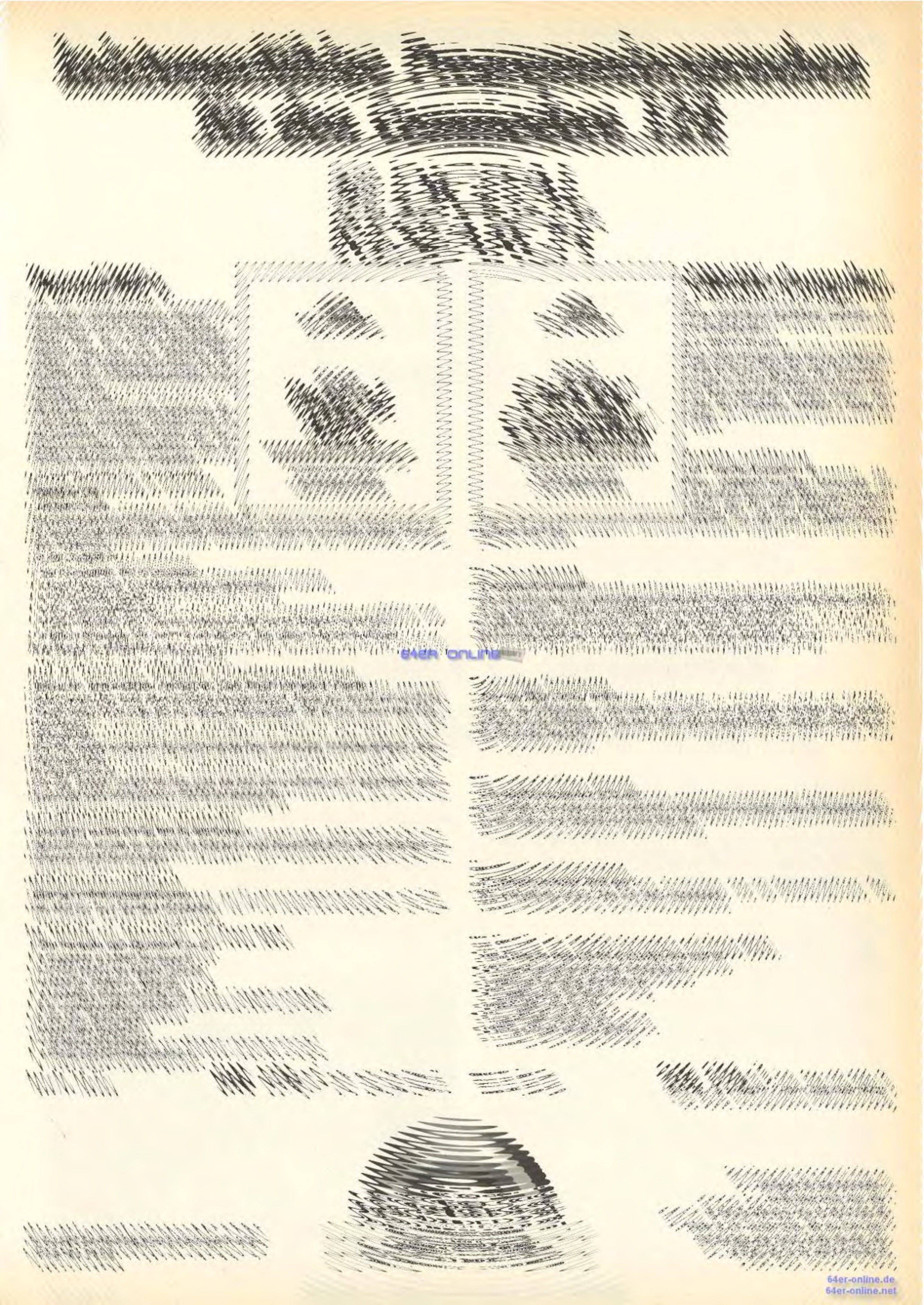


Bild 5. In dieser Konfiguration ist der Zeichensatz von Basic aus lesbar (16 KByte ROM, 44 KByte RAM)

Bild 6. 52 KByte RAM, 4 KByte Zeichensatz und 8 KByte ROM (Kernel) aktiv (Basic-ROM ausgeblendet)

Bild 7. Neben dem Zeichensatz sind 60 KByte RAM eingeblendet. Der I/O-Bereich kann nicht genutzt werden

Bild 8. Gleiche Speicherkonfiguration wie in Bild 4. Ausschließlich 64 KByte RAM (verfügbar)



Spalten
1. Spalte
23. Spalte
36. Spalte
40. Spalte
1. Zeile
5. Zeile
12. Zeile
25. Zeile
Zeilen
obere Grenze
untere Grenze
linke Grenze
rechte Grenze
Window

Bild 1. Schematische Darstellung eines Windows

Windows – Fenster zum neuen Bedienungskomfort

Der Begriff »Window« hat vor allem durch neuere Computer, etwa dem AMIGA, einen größeren Bekanntheitsgrad erreicht. In diesem Kurs erfahren Sie vieles über die »Window-Technik« sowie deren Programmierung und Simulation auf Ihrem C16, C64 oder C128.

Zunächst einmal wollen wir klären, was ein Window ist. Es handelt sich dabei um einen Bildschirmausschnitt, der als Teilbildschirm eingeblendet und separat, das heißt vom Hauptbildschirm unabhängig, behandelt wird. Die Ausgabe eines Windows geschieht durch Überschreiben eines Teils des Gesamtbildschirms mit dem Window-Text.

Dadurch, daß der alte Bildschirm nicht gelöscht, sondern nur teilweise überlagert wird, ist der vorherige Bildschirminhalt noch zu einem beträchtlichen Teil erkennbar. Werden viele solche Überlagerungen durchgeführt, entsteht allerdings ein relativ chaotisches Aussehen des Bildschirms, da von einzelnen Windows nur noch Bruchstücke zu sehen sind. Grundvoraussetzung ist, daß das jeweils »aktuelle« Window in vollem Umfang eingeblendet ist.

Nach dieser kurzen Theorie wollen wir uns die Praxis ansehen. In Bild 1 sehen Sie die schematische Darstellung eines Windows. Die Größe des gesamten Bildschirms setzt sich aus 40 Zeichen mal 25 Zeilen (siehe Bild 1) beim C16, C64 und C128 im 40-Zeichen-Modus zusammen, also insgesamt 1000 Zeichen. In Bild 1 sehen Sie, was ein Window ist: Ein Teilbereich von der 23. bis zur 36. Spalte und von der 5. bis zur 12. Zeile wird als Window definiert und bekommt eine bestimmte Aufgabe zugewiesen. Zum Beispiel könnten wir programmieren, daß dort dauernd die Uhrzeit eingeblendet wird oder es könnte ein Auswahlmenü erscheinen. Es gibt viele Anwendungsmöglichkeiten!

Um Ihnen einen Eindruck zu geben, wie ein kommerzielles Windowprogramm aussehen kann, zeigen wir Ihnen einige Beispiele anhand des Amigas: Bild 2 wurde von der AMIGA-

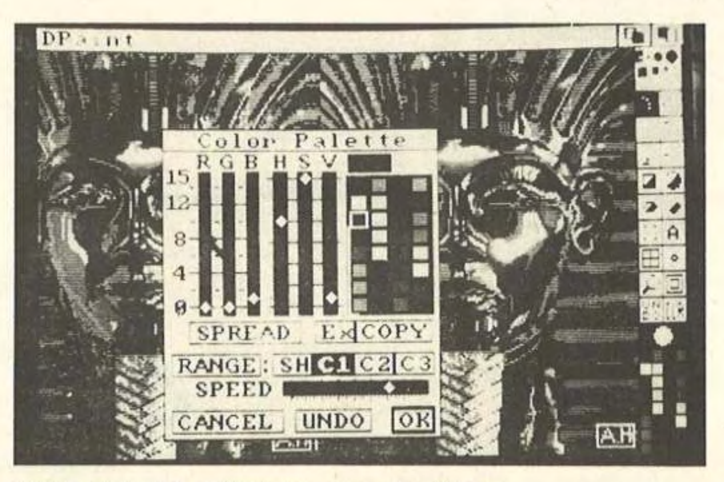


Bild 2. Beispiel: AMIGA-Benutzeroberfläche

Benutzeroberfläche »Intuition«, Bild 3 von der C64-Benutzeroberfläche GEOS (Testbericht siehe 64'er, Ausgabe 6/86) und Bild 4 vom C128-Textverarbeitungsprogramm Vizawrite Classic (Test: 64'er, 5/86) aufgenommen.

Das erste Window-Programm

Nehmen wir wieder Bild 1 zur Hand. Wenn wir fürs erste keine größeren Ansprüche stellen, als daß der in Bild 1 als Window definierte Bereich unabhängig vom Gesamtbildschirm beschrieben wird, genügt uns Listing 1. Dieses Programm läuft auf C16, C64 und auf dem C128 im 40-Zeichen-Modus. Wenn Sie es ganz normal über »RUN« starten, wird zunächst der Bildschirm mit zufälligen Zeichen beschrieben. Dies erleichtert die Demonstration, daß nur der Bildschirmbereich des Windows genutzt wird und dieser eigenständig ist. Dann wird das Window ausgegeben. Mit einem Tastendruck wird das Programm beendet.

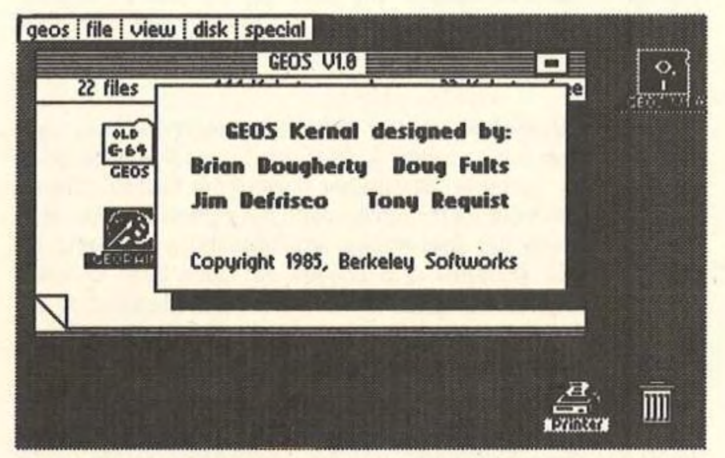


Bild 3. Beispiel: Geos-Benutzeroberfläche des C64

In diesem Anwendungsbeispiel wird das Window rein demonstrativ genutzt, es zeigt lediglich seine »persönlichen« Daten an. Rein von der Betrachtung des laufenden Programms kann man auch sehen, daß das Window optisch durch eine Umrahmung hervorgehoben wird. Diese Eingrenzung ist bei Commodore-Computern dank der Grafikzeichen einfach realisierbar, denn Kasten- und Rahmen-Symbole sind in ausreichender Menge vorhanden. Die grafische Hervorhebung ist von elementarer Bedeutung, damit man klar ersehen kann, wo ein Window beginnt und wo es endet, denn Windows sind keine Spielerei, sondern sollen statt dessen eine ökonomische Nutzung des Bildschirms ermöglichen.

Bei einem Start des Programms über GOTO 360 wird das Überschreiben des Bildschirms durch Zufallszahlen übersprungen. Sie könnten zum Beispiel das Programm über LIST auf dem Bildschirm ausgeben und dann ohne vorheriges Löschen des Bildschirms mit GOTO 360 starten. In jedem Fall wird das Window eingeblendet, egal was vorher auf dem Bildschirm stand. Genau diese Eigenschaft macht die Flexibilität der Windows aus.

Die Programmiertechnik

Bis jetzt haben wir uns sehr wenig mit der Technik beschäftigt und das Ganze eher aus Anwendersicht betrachtet. Sie werden aber sehen, daß man keine besonderen Programmierkenntnisse braucht, um Windows zu realisieren. Im Prinzip ist es nur eine Frage der Planung des Programms und im besonderen der Bildschirmausgabe. Bei neueren Computern wird eine ausgereifte Window-Technik bereits vom Betriebssystem unterstützt; deswegen brauchen wir aber noch lange nicht zu verzweifeln, denn vieles, was die Großen können, schaffen die Kleinen auch. In diesem Zusammenhang möchte ich gleich erwähnen, daß die sogenannte »Window-Technik« von C16 und C128 für unsere Zwecke unbrauchbar ist, denn es handelt sich nur um eine Verkleinerung des Bildschirms.

Kommen wir zurück zur Programmierung. Weil die Ausgabe-Routinen des Computers nicht in der Lage sind, für uns darauf zu achten, daß wir nicht über die Grenzen eines Windows hinausschreiben, müssen wir solche Vorkehrungen treffen. Als erstes müssen wir zur Ausgabe eines Windows auf die richtige Positionierung des Cursors achten. In unserem Beispiel (wir befinden uns immer noch bei Listing 1) muß der Cursor in die 5. Zeile bewegt werden, da der obere Rand des Windows mit 5 festgelegt ist. Dieses Ansteuern der 5. Zeile erreichen wir in den Zeilen 390 bis 420. Der Code CHR$(17) in Zeile 420 steht für <CURSOR-DOWN>. Dadurch, daß in Zeile 420 insgesamt 4mal der Cursor von der obersten Zeile aus nach unten bewegt wird, landen wir logischerweise in der 5. Zeile. Falls es Ihnen nicht klar sein sollte, probieren Sie es doch im Basic-Editor mit der CURSOR-DOWN- und HOME-Taste aus.

Damit hätten wir schon die obere Grenze eingehalten. Von gleicher Wichtigkeit ist die linke Grenze. Wir können nicht ganz links am Bildschirmrand mit der Ausgabe einer Zeile beginnen, sondern erst in der 23. Spalte. Dies heißt für uns, daß wir die ersten 22 Spalten einer Bildschirmzeile überspringen müssen, wofür die CURSOR-RIGHT-Bewegung verantwortlich ist (siehe zum Beispiel Zeile 460).

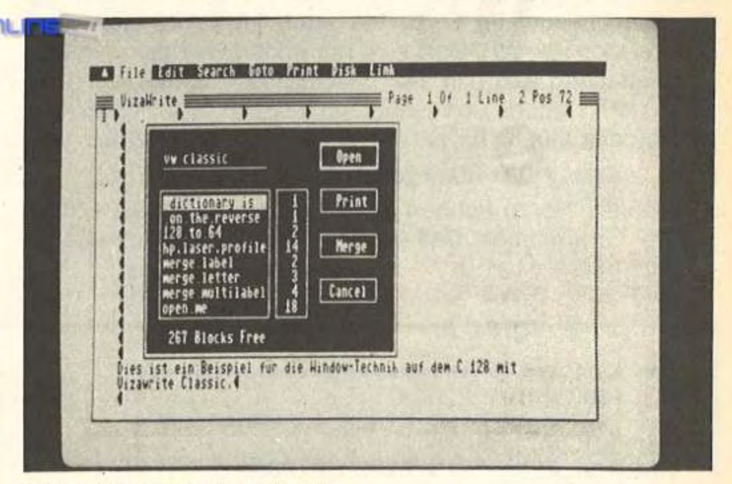


Bild 4. Vizawrite Classic 128

Wenn wir dann eine richtige Position am Bildschirm angesteuert haben, kann eine Zeile des Windows ausgegeben werden. Es ist unbedingt darauf zu achten, daß der Text nicht länger, als das Window breit ist. Die Breite des Windows ergibt sich aus linkem und rechtem Rand. In unserem Fall ist der linke Rand die 23. der rechte Rand die 36. Spalte. Die Breite beträgt 36-23+1 = 14 Zeichen. Die 1 muß deshalb addiert werden, da auch die Endspalten 23 und 36 genutzt werden. An den Zeilen 460 bis 530 können Sie sehen, daß nach den CURSOR-RIGHT-Steuerzeichen nur 14 Zeichen ausgegeben werden – nicht mehr und nicht weniger.

Nach der Ausgabe der Window-Zeile muß die nächste Zeile angesteuert werden. Hierzu füllen wir den Rest der Zeile mit <CURSOR-RIGHT>-Befehlen. In unserem Beispiel sind dies 4, denn von der Bildschirmbreite (40 Zeichen) muß einfach die rechte Grenze (36. Spalte) subtrahiert werden: 40 - 36 = 4.

Am Ende der Zeile steht hinter den Anführungszeichen ein Semikolon, der einen Zeilenvorschub verhindert. Wenn wir keine Steuerzeichen außer CURSOR RIGHT verwenden, hat jetzt ein Ausgabestring (siehe Zeile 460) genau 40 Zeichen Länge. Dies geht bis zur letzten Window-Zeile (Zeile 530); da nach der letzten Window-Zeile nicht mehr ins Window gedruckt werden soll, sind die vier CURSOR RIGHTs am Ende überflüssig und werden auch nicht ausgegeben. Der Strichpunkt ist aber weiterhin erforderlich, denn ein Zeilenvorschub kann den Bildschirmaufbau sehr durcheinander bringen (besonders beim C 16/C 128!), wenn der Computer beispielsweise ein Scrolling unternimmt. Dies können Sie gut beobachten, wenn Sie das Programm per Tastendruck beenden.

Besser mit dem CHAR-Befehl

Sie sollten nun ein wenig an Listing 1 herumexperimentieren. Falls Sie keinen C 64, sondern einen C 16/C 128 haben, ist Listing 1 für Sie von nun an uninteressant, denn beim C 16/C 128 gibt es eine viel einfachere Lösung unter Einsatz des CHAR-Befehls. Dieser ermöglicht ein Ausgeben an fest vorgegebene Cursorpositionen, so daß wir die beim C 64 erforderlichen Cursorpositionierungen über Steuerzeichen vergessen können.

Listing 2 hat denselben Effekt wie Listing 1, ist allerdings wesentlich kürzer, da die Steuerzeichen wegfallen. Geändert hat sich, daß hier das CHAR-Kommando, welches das Basic 2,0 des C 64 nicht kennt, eingesetzt wird und somit die Cursorpositionierungen aus Listing 1 entfallen. Wie nun dieser CHAR-Befehl verwendet wird, bedarf einiger Erläuterung, die man sowohl im C 16- als auch im C 128-Handbuch schmerzlich vermißt. Dort wird nur erklärt, wie man mit dem CHAR-Befehl Texte in hochauflösender Grafik schreibt. CHAR kann jedoch noch mehr, es ist auch eine Ausgabe im Text-Modus möglich. Dann lautet die Syntax wie folgt:

```
CHAR,Spalte,Zeile,"Ausgabetext als String"(,1)
```

Beispiele hierzu können Sie den Zeilen 460 bis 530 in Listing 2 entnehmen. Daß auf den CHAR-Befehl unmittelbar ein Komma folgt, ist kein Druckfehler; damit soll dem Computer angezeigt werden, daß eine Ausgabe in den Textbildschirm gewünscht ist. Statt »CHAR,...« könnte man auch »CHAR1,...« schreiben, erübrigt sich aber, da es im Betriebssystem bereits voreingestellt ist. Der CHAR-Befehl rechnet die Spalten und Zeilen nicht von 1 bis 40 beziehungsweise 1 bis 25, wie wir es bislang getan haben, sondern von 0 bis 39 beziehungsweise 0 bis 24. Da wir Menschen aber beim Zählen mit 1 beginnen, müssen wir uns geringfügig umstellen. Der Ausgabestring enthält nur den Text, der in das Window gehört. Die fehlenden Steuerzeichen-Orgien tragen sehr zur Übersichtlichkeit und Kürze der Listings bei. Um einem Trugschluß vorzubeugen: Der Ausgabestring des CHAR-Befehls wird nicht wie bei PRINT angegeben, das heißt nach dem Ausgabestring darf kein Semikolon stehen, das Komma ist auch nicht erlaubt. Operationen wie »+«, »MID$«, »RIGHT$«, »LEFT$« oder »STR$« sind weiterhin zugelassen.

Beim CHAR-Befehl ist das Schreiben der Window-Zeilen leichter, wenn man darauf achtet, daß der Ausgabestring jeweils in der gleichen Spalte am Bildschirm beim Editieren beginnt. Dann bekommt man nämlich eine genauere Vorstellung davon, wie der Text später am Bildschirm aussieht. Da die Parameter »Spalte« und »Zeile« ein oder zwei Zeichen Länge haben, könnte es leicht sein, daß beim Listen die Ausgabestrings nicht genau, sondern leicht verschoben untereinander stehen. Dies kann man verhindern, indem man bei einstelligen Parametern vor den Parameter ein Leerzeichen setzt: in Zeile 460 wird vor die einstellige Zahl »4« (Zeilenangabe) ein Leerzeichen gesetzt, damit die Zeile genausolang ist wie zum Beispiel Zeile 520, in der der Parameter Zeile schon zweistellig ist. Dieses Einfügen von Leerzeichen ist für die Funktion der Zeile nicht erforderlich, denn Leerzeichen außerhalb von Anführungszeichen überliest der Computer bekanntlich; es ist aber dennoch kein übersteigertes Schönheitsbewußtsein, sondern – wie gesagt – eine Hilfe beim Schreiben des Ausgabestrings.

Die Window-Ausgabe-Zeilen mit dem CHAR-Befehl müssen immer dieselbe Zahl als Spaltenangabe haben; die Zeilenangabe wird von Zeile zu Zeile um 1 erhöht (siehe Zeilen

```
100 REM *****************************
110 REM *                           *
120 REM * ANZEIGE EINER INFORMATION *
130 REM *                           *
140 REM *       IN EINEM WINDOW     *
150 REM *                           *
160 REM *****************************
170 REM *                           *
180 REM * START MIT ZUFAELLIGEM BE- *
190 REM * SCHREIBEN DES BILDSCHIRMS *
200 REM * ZUR HERVORHEBUNG DES WIN- *
210 REM * DOWS: 'RUN' ODER 'GOTO300' *
220 REM *                           *
230 REM * START OHNE AENDERUNG DES  *
240 REM * BILDSCHIRMS: GOTO 360     *
250 REM *                           *
260 REM * C16,C64,C128(40-Z.-MODUS) *
270 REM *                           *
280 REM *****************************
290 :
300 REM BILDSCHIRM MIT ZUFAELLIGEN
310 REM ZEICHEN BESCHREIBEN:
320 :
330 PRINT CHR$(147);: REM BILDSCHIRM LOESCHE
    N
340 FOR F=1 TO 950: PRINT CHR$(RND(0)*48+48)
    ;: NEXT F: REM 950 ZUFAELLIGE ZEICHEN
350 :
360 REM WINDOW AUSGEBEN
370 REM ===============
380 :
390 REM CURSOR IN 5. ZEILE BEWEGEN
400 :
410 PRINT "{HOME}";: REM 'CURSOR HOME' AUSFU
    EHREN
420 FOR F=1 TO 4: PRINT CHR$(17);: NEXT F: R
    EM 4 ZEILEN UEBERSPRINGEN
430 :
440 REM NUN WIRD DAS WINDOW AUSGEGEBEN:
450 :
460 PRINT "{22RIGHT}╤************╕{4RIGHT}";
470 PRINT "{22RIGHT}=>> WINDOW <<={4RIGHT}";
480 PRINT "{22RIGHT}╓************╖{4RIGHT}";
490 PRINT "{22RIGHT}=LINKE GR.:23={4RIGHT}";
500 PRINT "{22RIGHT}=RECHTE G.:36={4RIGHT}";
510 PRINT "{22RIGHT}=OBERE GR.: 5={4RIGHT}";
520 PRINT "{22RIGHT}=UNTERE G.:12={4RIGHT}";
530 PRINT "{22RIGHT}╘************╛";
540 :
550 GET A$: IF A$="" THEN 550
```

Listing 1. Window-Beispiel für den C16, C64, C128 im 40-Zeichen-Modus

460 bis 530), denn sonst würden wir immer nur dieselbe Window-Zeile drucken.

Der CHAR-Befehl bietet noch einen Vorteil gegenüber dem PRINT-Kommando, der für die C128-Besitzer ziemlich wichtig ist: Ein mit CHAR-Befehlen arbeitendes Programm läuft auch im 80-Zeichen-Modus, wie Sie im Vergleich zwischen Listing 1 und Listing 2 erkennen können: Listing 1 (arbeitet mit PRINT) ist nur im 40-Zeichen-Modus funktionsfähig und müßte für den 80-Zeichen-Modus angepaßt werden, Listing 2 (verwendet die CHAR-Anweisung) läuft in beiden Modi auch ohne Umschreibung.

Windows im 80-Zeichen-Bildschirm des C128

Da ein Window wie jeder andere Text auch Platz auf dem Bildschirm beansprucht, kann man auf einem größeren Bildschirm mehr und größere Windows unterbringen als auf einem kleinen. Deshalb eignet sich der 80-Zeichen-Bildschirm des C128 besonders, denn er faßt doppelt soviele als der 40er-Bildschirm. Wir wollen uns diese größere Kapazität zunutze machen und lassen ein wesentlich größeres Window ausgeben. Listing 3 setzt wieder den CHAR-Befehl ein. In Zeile 320 wird der 80-Zeichen-Modus eingeschaltet und der Prozessor auf doppelte Geschwindigkeit umgestellt. Beim zufälligen »Vollschreiben« des Bildschirms in Zeile 340 werden nun 1999 (statt vorher 999) Zeichen ausgegeben, denn der 80-Zeichen-Bildschirm faßt ja 1000 Zeichen mehr als der 40er.

Mehrere Windows

Bislang haben wir jeweils ein einziges Window ausgegeben. Listing 4 ist nun ein Anwendungsbeispiel, das sich mehrerer Windows bedient. Es handelt sich um ein Menü, wie es vor einem fiktiven Spielprogramm stehen könnte. Da Listing 4 auch auf dem C64 laufen soll, wird auf den CHAR-Befehl verzichtet. Um die Cursorpositionierung trotzdem einfacher

```
100 REM ******************************
110 REM *                            *
120 REM * ANZEIGE EINER INFORMATION *
130 REM *                            *
140 REM *      IN EINEM WINDOW       *
150 REM *                            *
160 REM ******************************
170 REM *                            *
180 REM * START MIT ZUFAELLIGEM BE- *
190 REM * SCHREIBEN DES BILDSCHIRMS *
200 REM * ZUR HERVORHEBUNG DES WIN- *
210 REM * DOWS: 'RUN' ODER 'GOTO300' *
220 REM *                            *
230 REM * START OHNE AENDERUNG DES   *
240 REM * BILDSCHIRMS: GOTO 360      *
250 REM *                            *
260 REM *        C16,C128            *
270 REM *                            *
280 REM ******************************
290 :
300 REM BILDSCHIRM MIT ZUFAELLIGEN
310 REM ZEICHEN BESCHREIBEN:
320 :
330 PRINT CHR$(147);: REM BILDSCHIRM LOESCHE
    N
340 FOR F=1 TO 950: PRINT CHR$(RND(0)*48+48)
    ;: NEXT F: REM 950 ZUFAELLIGE ZEICHEN
350 :
360 REM WINDOW AUSGEBEN
370 REM ==============
380 :
460 CHAR ,22, 4,"R**************S"
470 CHAR ,22, 5,"=>> WINDOW <<="
480 CHAR ,22, 6,"Q**************W"
490 CHAR ,22, 7,"-LINKE GR.:23="
500 CHAR ,22, 8,"-RECHTE G.:36="
510 CHAR ,22, 9,"-OBERE GR.: 5="
520 CHAR ,22,10,"-UNTERE G.:12="
530 CHAR ,22,11,"Z**************X"
540 :
550 GET A$: IF A$="" THEN 550
```

Listing 2. Window-Beispiel für den C16, C128 im 40-Zeichen-Modus

```
100 REM ******************************
110 REM *                            *
120 REM * ANZEIGE EINER INFORMATION *
130 REM *                            *
140 REM *      IN EINEM WINDOW       *
150 REM *                            *
160 REM ******************************
170 REM *                            *
180 REM * START MIT ZUFAELLIGEM BE- *
190 REM * SCHREIBEN DES BILDSCHIRMS *
200 REM * ZUR HERVORHEBUNG DES WIN- *
210 REM * DOWS: 'RUN' ODER 'GOTO300' *
220 REM *                            *
230 REM * START OHNE AENDERUNG DES   *
240 REM * BILDSCHIRMS: GOTO 360      *
250 REM *                            *
260 REM * C128  IM 80-ZEICHEN-MODUS *
270 REM *                            *
280 REM ******************************
290 :
300 REM BILDSCHIRM MIT ZUFAELLIGEN
310 REM ZEICHEN BESCHREIBEN:
320 GRAPHIC 5: FAST
330 PRINT CHR$(147);: REM BILDSCHIRM LOESCHE
    N
340 FOR F=1 TO 1999: PRINT CHR$(RND(0)*48+48
    );: NEXT F: REM 1999 ZUFAELLIGE ZEICHEN
350 :
360 REM WINDOW AUSGEBEN
370 REM ==============
380 :
460 CHAR ,22, 4,"R**********************
    ********S"
470 CHAR ,22, 5,"=>>>> ETWAS GROESSERES WIND
    OW <<<<="
480 CHAR ,22, 6,"Q*************R********
    ********W"
490 CHAR ,22, 7,"-LINKE GRENZE:23-RECHTE GRE
    NZE:57 ="
500 CHAR ,22, 8,"-OBERE GRENZE: 5-UNTERE GRE
    NZE:17 ="
510 CHAR ,22, 9,"Q*************R********
    ********W"
520 CHAR ,22,10,"-DA DER 80-ZEICHEN-BILDSCHI
    RM DOP-="
530 CHAR ,22,11,"-PELT SOVIELE ZEICHEN AUFNE
    HMEN{3SPACE}="
540 CHAR ,22,12,"-KANN WIE DER 40'ER, SIND A
    UCH{4SPACE}="
550 CHAR ,22,13,"-SOLCH GROSSE WINDOWS, DIE
    SONST{2SPACE}="
560 CHAR ,22,14,"-ZUVIEL PLATZ AM BILDSCHIRM
     BEAN- ="
570 CHAR ,22,15,"-SPRUCHEN WUERDEN, MOEGLICH
    .{6SPACE}="
580 CHAR ,22,16,"Z**********************
    ********X"
590 :
600 GET KEY A$
```

Listing 3. Window-Beispiel für den C128 im 80-Zeichen-Modus

zu machen, wird das CURSOR-DOWN-Steuerzeichen jeweils in entsprechender Häufigkeit nach dem HOME-Steuerzeichen ausgegeben (siehe Zeilen 340, 440, 510 und 580).

Durch Drücken der Tasten 1 bis 3 wird der Wert für Geschwindigkeit, Anzahl der Spieler oder Spielstufe erhöht; wird der höchste erlaubte Wert überschritten, so wird wieder 1 eingestellt; 4 beendet das Programm.

An Listing 4 kann man einige verschiedene Möglichkeiten der optischen Eingrenzung eines Windows erkennen.

Durch Kommentare in »REM«-Zeilen erklärt sich der Programmaufbau von selbst.

Text unter Window retten

Einen großen Nachteil haben alle bisher von uns ausgegebenen Windows gehabt: Der Text, der vorher an der Position des Windows stand, ist verlorengegangen, da er durch das Window überschrieben wurde. Dies macht in 50 Prozent aller Fälle nichts aus, aber was ist mit den anderen 50 Prozent?

Wir wollen den unter dem Window liegenden Bildschirmbereich retten, indem wir ihn über PEEK auslesen und in einem Variablenarray speichern. Wenn das Window verschwunden und der alte Text eingeblendet werden soll, wird das Variablenarray wieder in den Bildschirmspeicher geschrieben. Diese Anforderung erfüllen Listing 5 und 6, die wir zunächst nur ausprobieren, dann aber eingehend besprechen wollen.

Listing 5 ist die C 64/C 128-Version (C 128 im 40-Zeichen-Modus!), Listing 6 eine C 16-Fassung. Die C 16-Version unterscheidet sich von Listing 5 nur durch den Einsatz des CHAR-Befehls und eine andere Basisadresse des Bildschirmspeichers. Wenn wir das Programm starten (C 64/C 128: Listing 5 / C 16: Listing 6), wartet das Programm auf einen Tastendruck. »N« bewirkt, daß der Bildschirm nicht zur Demonstration mit zufälligen Zeichen überschrieben wird; eine beliebige andere Taste ruft das »Vollschreiben« hervor.

Dann wird das Window, das Sie aus den Listings 1 und 2 kennen, ausgegeben. Wenn Sie eine Taste drücken, wird der vorher an der Windowposition stehende Text angezeigt, ein weiterer Tastendruck blendet wieder das Window ein und so weiter.

Befassen wir uns nun mit der Programmierung. In Zeile 270 wird jeweils ein 111+1 = 112 Elemente großes Array T(n) dimensioniert. Die »+1« ergibt sich daraus, das die Zählung wieder mit Null beginnt. Zwischen Null und 111 liegen 112 Elemente (T(0)...T(111)). In diesem Array werden später die »unter dem Window liegenden« Zeichen gespeichert. Die Größe ergibt sich aus der Anzahl der Spalten des Windows multipliziert mit der Anzahl der Zeilen des Windows: 14 * 8 = 112.

Der an der Windowposition liegende Text muß noch vor der Ausgabe des Windows gerettet werden, da er ja durch das Window überschrieben wird. Dies erreichen wir in den Zeilen 420 bis 460. Wie Sie sehen, muß auch dort die computerinterne Numerierung, die jeweils um 1 kleiner ist als die, die wir gewohnt sind, verwendet werden; deshalb heißt es zum Beispiel in Zeile 430 »4 TO 11«, gemeint sind aber – wie der REM-Kommentar dahinter sagt – die Zeilen 5 bis 12 nach der mit 1 beginnenden Zählweise.

In Listing 5 lesen die Zeilen 650 bis 690 den Text wieder aus dem Array T(n) ein und schreiben ihn in den Bildschirmspeicher, in Listing 6 wird dies durch die Zeilen 640 bis 680 erreicht.


```
100 REM ********************************
110 REM *                              *
120 REM * B E I S P I E L - M E N U E *
130 REM *                              *
140 REM *      MIT WINDOW-TECHNIK      *
150 REM *                              *
160 REM *  C16,C64,C128(40-Z.-MODUS)   *
170 REM *                              *
180 REM ********************************
190 :
200 L=1: G=1: A=1: REM LEVEL, GESCHWINDIGKEI
    T UND ANZAHL DER SPIELER VORBELEGEN
210 PRINT CHR$(147);
220 PRINT TAB(7)"BEISPIELMENUE FUER WINDOWS"
230 PRINT TAB(7)"--------------------------"
240 PRINT "{5DOWN}"
250 PRINT TAB(3);CHR$(18)" 1 " CHR$(146);TAB
    (10);"SPIELSTUFE{DOWN}"
260 PRINT TAB(3);CHR$(18)" 2 " CHR$(146);TAB
    (10);"GESCHWINDIGKEIT{DOWN}"
270 PRINT TAB(3);CHR$(18)" 3 " CHR$(146);TAB
    (10);"ANZAHL DER SPIELER{DOWN}"
280 PRINT TAB(3);CHR$(18)" 4 " CHR$(146);TAB
    (10);"AUSWAHL BEENDEN{2DOWN}"
290 REM NACH DEN NORMALEN PRINT-ZEILEN  WERD
    EN JETZT DIE WINDOWS GEDRUCKT
300 :
310 REM ZUERST DAS COPYRIGHT-WINDOW:
320 REM ===========================
330 :
340 PRINT "{HOME,2DOWN}";
350 PRINT "{5RIGHT}U************I{24RIGHT}";
360 PRINT "{5RIGHT}-COPYRIGHT-{24RIGHT}";
370 PRINT "{5RIGHT}- WINDOW: -{24RIGHT}";
380 PRINT "{5RIGHT}Q************W{24RIGHT}";
390 PRINT "{5RIGHT}-(C) 64'ER-{24RIGHT}";
400 PRINT "{5RIGHT}J************K"
410 :
420 REM NUN DAS LEVEL-WINDOW:
430 REM ====================
440 PRINT "{HOME,5DOWN}";
450 PRINT "{22RIGHT,RVSON,15SPACE,3RIGHT}";
460 PRINT "{22RIGHT,RVSON} SPIELSTUFE:"+STR$
    (L)+" {3RIGHT}";
470 PRINT "{22RIGHT,RVSON,15SPACE,RVOFF}"
480 :
490 REM UND DAS GESCHWINDIGKEITSWINDOW:
500 REM ==============================
510 PRINT "{HOME,10DOWN}";
520 PRINT "{28RIGHT}R***********E{RIGHT}";
530 PRINT "{28RIGHT}-TEMPO:"+STR$(G)+" -{RIG
    HT}";
540 PRINT "{28RIGHT}Z***********X"
550 :
560 REM UND DAS SPIELERZAHL-WINDOW:
570 REM ==========================
580 PRINT "{HOME,16DOWN}";
590 PRINT "{25RIGHT}**************{2RIGHT}";
600 PRINT "{25RIGHT}*"+STR$(A)+" SPIELER *{2
    RIGHT}";
610 PRINT "{25RIGHT}**************{DOWN}"+CHR
    $(13)
620 :
630 :
640 PRINT "BITTE ENTSPRECHENDE TASTE DRUECKE
    N !"
650 :
660 GET A$: IF A$="" THEN 660
670 IF A$ = "1" THEN L=L+1: IF L>3 THEN L=1
680 IF A$ = "2" THEN G=G+1: IF G>3 THEN G=1
690 IF A$ = "3" THEN A=A+1: IF A>2 THEN A=1
700 IF A$ = "4" THEN END
710 :
720 GOTO 290
```

Listing 4. Anwendungsbeispiel für mehrere Windows

```
100 REM ********************************
110 REM *                              *
120 REM *    ANZEIGE EINES WINDOWS     *
130 REM *                              *
140 REM * UNTER BERUECKSICHTIGUNG DES  *
150 REM *                              *
160 REM * UNTER DEM WINDOW LIEGENDEN   *
170 REM *                              *
180 REM *     BILDSCHIRMINHALTES       *
190 REM *                              *
200 REM ********************************
210 REM *                              *
220 REM *  C64 UND C128 (40 ZEICHEN)   *
230 REM *                              *
240 REM ********************************
250 :
260 B = 1024: REM BASISADRESSE DES BILDSCHIR
    MSPEICHERS
270 DIM T(111): REM ARRAY ZUM RETTEN DES TEX
    TES (S. 350-)
280 :
290 GET A$: IF A$="" THEN 290
300 IF A$="N" THEN 350: REM BEI "N" BILDSCHI
    RM NICHT ZUFAELLIG BESCHREIBEN
310 :
320 PRINT CHR$(147);
330 FOR F=1 TO 999: PRINT CHR$(35+80*RND(0))
    ;: NEXT
340 :
350 REM AN WINDOW-POSITIONEN LIEGENDEN
360 REM TEXT RETTEN
370 :
380 REM DAS WINDOW BEANSPRUCHT:
390 REM DIE SPALTEN 23-36
400 REM I.D. ZEILEN 5-12
410 :
420 I=0: REM MIT INDEX 0 BEGINNEN
430 FOR Z= 4 TO 11: REM ZEILEN : 5-12
440 FOR S=22 TO 35: REM SPALTEN: 23-36
450 T(I)=PEEK(B+40*Z+S): I=I+1: REM WERT EIN
    LESEN, INDEX ERHOEHEN
460 NEXT S,Z
470 :
480 REM NUN WIRD DAS WINDOW AUSGEGEBEN:
490 :
500 PRINT "{HOME,4DOWN}";
510 PRINT "{22RIGHT}R************S{4RIGHT}";
520 PRINT "{22RIGHT}=>> WINDOW <<={4RIGHT}";
530 PRINT "{22RIGHT}Q************W{4RIGHT}";
540 PRINT "{22RIGHT}=LINKE GR.:23={4RIGHT}";
550 PRINT "{22RIGHT}=RECHTE G.:36={4RIGHT}";
560 PRINT "{22RIGHT}=OBERE GR.: 5={4RIGHT}";
570 PRINT "{22RIGHT}=UNTERE G.:12={4RIGHT}";
580 PRINT "{22RIGHT}Z************X";
590 :
600 GET A$: IF A$="" THEN 600
610 :
620 REM NUN WIRD DER TEXT UNTER DEM
630 REM WINDOW WIEDER ANGEZEIGT:
640 :
650 I=0
660 FOR Z= 4 TO 11
670 FOR S=22 TO 35
680 POKE B+40*Z+S,T(I): I=I+1: REM WERT SCHR
    EIBEN, INDEX ERHOEHEN
690 NEXT S,Z
700 :
710 GET A$: IF A$="" THEN 710
720 :
730 GOTO 350: REM WIEDER RETTEN UND WINDOW A
    NZEIGEN
```

Listing 5. Window-Technik mit Rettung des Bildschirminhaltes auf dem C64 und C128 im 40-Zeichen-Modus

```
100 REM ********************************
110 REM *                              *
120 REM *    ANZEIGE EINES WINDOWS     *
130 REM *                              *
140 REM * UNTER BERUECKSICHTIGUNG DES  *
150 REM *                              *
160 REM * UNTER DEM WINDOW LIEGENDEN   *
170 REM *                              *
180 REM *     BILDSCHIRMINHALTES       *
190 REM *                              *
200 REM ********************************
210 REM *                              *
220 REM *          C16/116             *
230 REM *                              *
240 REM ********************************
250 :
260 B = 3072: REM BASISADRESSE DES BILDSCHIR
    MSPEICHERS BEIM C16
270 DIM T(111): REM ARRAY ZUM RETTEN DES TEX
    TES (S. 350-)
280 :
290 GET A$: IF A$="" THEN 290
300 IF A$="N" THEN 350: REM BEI "N" BILDSCHI
    RM NICHT ZUFAELLIG BESCHREIBEN
310 :
320 PRINT CHR$(147);
330 FOR F=1 TO 999: PRINT CHR$(35+80*RND(0))
    ;: NEXT
340 :
350 REM AN WINDOW-POSITIONEN LIEGENDEN
360 REM TEXT RETTEN
370 :
380 REM DAS WINDOW BEANSPRUCHT:
390 REM DIE SPALTEN 23-36
400 REM I.D. ZEILEN 5-12
410 :
420 I=0: REM MIT INDEX 0 BEGINNEN
430 FOR Z= 4 TO 11: REM ZEILEN : 5-12
440 FOR S=22 TO 35: REM SPALTEN: 23-36
450 T(I)=PEEK(B+40*Z+S): I=I+1: REM WERT EIN
    LESEN, INDEX ERHOEHEN
460 NEXT S,Z
470 :
480 REM NUN WIRD DAS WINDOW AUSGEGEBEN:
490 :
500 CHAR ,22, 4,"R************S"
510 CHAR ,22, 5,"=>> WINDOW <<="
520 CHAR ,22, 6,"Q************W"
530 CHAR ,22, 7,"=LINKE GR.:23="
540 CHAR ,22, 8,"=RECHTE G.:36="
550 CHAR ,22, 9,"=OBERE GR.: 5="
560 CHAR ,22,10,"=UNTERE G.:12="
570 CHAR ,22,11,"Z************X"
580 :
590 GET A$: IF A$="" THEN 590
600 :
610 REM NUN WIRD DER TEXT UNTER DEM
620 REM WINDOW WIEDER ANGEZEIGT:
630 :
640 I=0
650 FOR Z= 4 TO 11
660 FOR S=22 TO 35
670 POKE B+40*Z+S,T(I): I=I+1: REM WERT SCHR
    EIBEN, INDEX ERHOEHEN
680 NEXT S,Z
690 :
700 GET A$: IF A$="" THEN 700
710 :
720 GOTO 350: REM WIEDER RETTEN UND WINDOW A
    NZEIGEN
```

Listing 6. Window-Technik mit Rettung des Bildschirminhaltes auf dem C16

Nachdem wir jetzt alle Grundlagen der Window-Programmierung in Basic erarbeitet haben, soll auch die Programmierung in Maschinensprache behandelt werden. Damit werden wir uns bis zum Ende dieses Kurses befassen. Sollte Ihnen etwas nicht klar sein, so können Sie durch das Experimentieren an den Listings ein wenig Erfahrung sammeln.

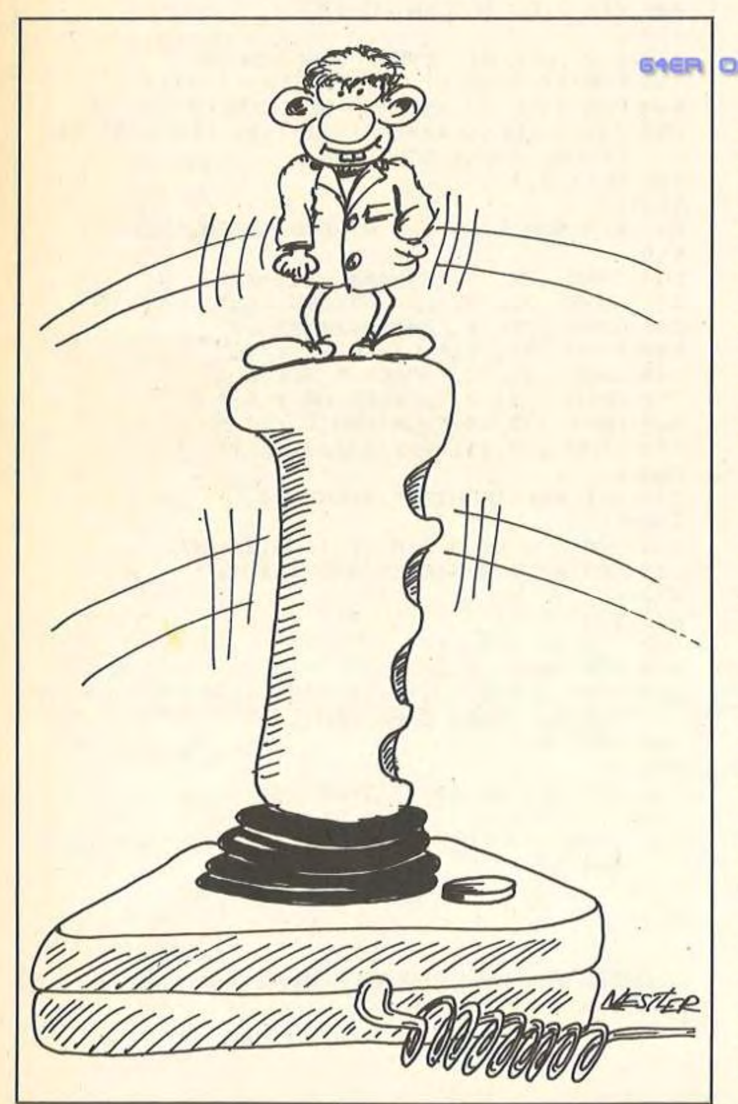
Windows in Maschinensprache

Unser Ziel ist nun, Maschinenroutinen für Windows zu schreiben, die wir in ein Demoprogramm einbinden wollen. Wir werden zwei Routinen entwickeln; die eine nennen wir OUTW, die andere PRWIN.

OUTW soll ähnlich wie die Betriebssystemroutine BASOUT ($FFD2) funktionieren, nur daß das auszugebende Zeichen in das Window und nicht in den Normalbildschirm geschrieben wird; PRWIN soll ein ganzes Window in einem Zug ausdrucken.

Für jedes Window schreiben wir der Einfachheit halber eine eigene OUTW- und eine eigene PRWIN-Routine (OUTW1/PRWIN1 für Window 1, OUTW2/PRWIN2 für Window 2). Diese Routinen halten wir so, daß sie leicht an andere Windows angepaßt werden können.

Die fertige Lösung bekommen Sie in Form von Listing 7 (befindet sich auf der Leserservice-Diskette) vorgestellt. Dies ist ein C64-Quelltext vom Assembler Hypra-Ass. Eine C128-Entsprechung ist Listing 8; dieser Quelltext wurde mit dem Top-Ass erstellt, einem leistungsfähigen Assembler für den C128. Zur C16-Version kommen wir später.



Listing 7 und Listing 8 (befinden sich nur auf der Leserservice-Diskette) sind sogar in den Zeilennummern identisch, allerdings mußten einige Label und Assembler-Anweisungen auf den C128 umgeschrieben werden. Die Ähnlichkeit beider Listings kommt auch dadurch zustande, daß der Top-Ass eine C128-Weiterentwicklung des Hypra-Ass ist.

Zunächst wollen wir unser Demoprogramm einmal starten. Falls Sie einen C128 und Top-Ass haben, assemblieren Sie sich Listing 8. C64 Besitzer assemblieren sich Listing 7 mit Hypra-Ass. Der C128-Quelltext erzeugt beim Assemblieren auf Diskette den Objektcode unter dem Filenamen »O«. Mit BANK15:BOOT»O« können Sie diesen dann starten. Hypra-Ass assembliert Listing 7 direkt in den Speicher; mit SYS 49152 wird der Objektcode gestartet.

Auf der Programm-Diskette ist der fertige Objektcode gespeichert; dies erleichtert Ihnen das Abtippen oder das Umschreiben auf andere Assembler.

Nach dem Start erscheint in dem Window Nummer 1 das Menü. Mit <1> (Erklärung) wird eine Erklärung ein/ausgeblendet, mit <2> die Bildschirmfarbe geändert (beim C128 nur die Farbe des 40-Zeichen-Bildschirms!) und mit <3> das Programm beendet.

Die Erklärung erscheint (wie sollte es auch anders sein) in einem Window (Window #2).

Nun zum C16-Programm. Die C16-Version habe ich ebenfalls mit Hypra-Ass auf einem C64 erstellt (wie die C64-Version). Dazu müssen nur einige Zeilen geändert werden (Listing 9, befindet sich nur auf der Programm-Service-Diskette). Da wohl kaum einer sowohl C16 als auch C64 besitzt, ist Listing 10 (befindet sich nur auf der Programm-Service-Diskette) ein Hex-Dump des Objektcodes, den Sie mit dem im C16 eingebauten Monitor eingeben können. Das C16-Programm wird mit SYS 14848 gestartet.

Vorher sollten Sie es aber im Monitor mit

```
S "FILENAME",Gerät,3A00,3D68
```

speichern.

Befassen wir uns nun mit den Quelltexten; C64-Besitzer nehmen bitte Listing 7, die C128er Listing 8, C16-Fans Listing 7 und die Änderungen aus Listing 9 - Zeilenangaben haben immer für alle Listings Gültigkeit.

Zunächst werden die Symbole definiert. ZEILE und SPALTE sind Zeropage-Adressen, in denen die aktuelle Cursorzeile und -spalte stehen; da es sich um Betriebssystem-Variablen handelt, sind diese von C64 und C128 verschieden. Wir brauchen die Label ZEILE und SPALTE, um später die Cursorposition feststellen zu können.

ZEILE1/SPALTE1 beziehungsweise ZEILE2/SPALTE2 sind die Cursorpositionen innerhalb der Windows. Diese werden vom Programm laufend neu berechnet. SETCUR ist ein Betriebssystem-Einsprung zur Cursorpositionierung; im X-Register wird die Zeile, im Y-Register die Spalte übergeben. Dann wird der Cursor mit JSR SETCUR nach Wunsch positioniert.

BASOUT und GET sind Kernel-Routinen zur Aus- beziehungsweise Eingabe eines Zeichens. TEMP1 ist ein 2-Byte-Zwischenspeicher, in dem die PRWIN1/PRWIN2-Routine die Endadresse des auszugebenden Textes ablegt. ERKLFLAG ist ein Flag dafür, ob der Erklärungstext am Bildschirm steht (Inhalt von ERKLFLAG = 0) oder ob er nicht angezeigt ist (Inhalt von ERKLFLAG = 1). CLEAR, HOME und CR sind ASCII-Codes für einige Steuerzeichen.

LINKS1 / RECHTS1 / OBEN1 / UNTEN1 beziehungsweise LINKS2 / RECHTS2 / OBEN2 / UNTEN2 sind die Grenzen der Windows. LINKS1/LINKS2 und OBEN1/OBEN2 werden in der mit 0 beginnenden Numerierung angegeben. Zu RECHTS1/RECHTS2 und UNTEN1/UNTEN2 addieren wir noch 1 (siehe »+1« in 500, 520, 550, 570).

Aus diesen Parametern können wir dann in 590/600 die

Anzahl der Zeichen in einem Window berechnen und in LAENGE1 beziehungsweise LAENGE2 merken. Diese Länge ist für die PRWIN-Routinen wichtig, damit sofort feststeht, wieviele Zeichen ausgegeben werden müssen. OUTPUTBYTE (Zeile 620) ist ein Zwischenspeicher, in dem das über eine OUTW-Routine auszugebende Zeichen gespeichert wird.

TEXTPTR ist ein Zeropage-Zeiger für unsere Zwecke, den wir von der PRWIN-Routine auf das auszugebende Zeichen stellen lassen.

Mit diesen Erläuterungen und den Kommentaren im Quelltext dürfte es für Sie kein Problem sein, das Programm zu verstehen. Lediglich zu Zeile 4230 ist noch zu sagen, daß hier das ERKLFLAG von 0 auf 1 beziehungsweise von 1 auf 0 umgeschaltet wird.

Komfortable Windows

Damit Sie die Routinen auch für eigene Programme nutzen können, wollen wir die Parameterübergabe an OUTW/PRWIN besprechen, wie in einem Assembler-Kurs die Benutzung einer Betriebssystem-Routine besprochen wird.

Mit LDA #ASCII-Code des Zeichens
JSR OUTW ;
OUTW1 oder OUTW2

kann ein Zeichen ins Window ausgegeben werden. Nach dem Beenden der Routine bekommen Akku, X- und Y-Register denselben Wert wieder, den sie vor dem Aufruf hatten. Bei den ASCII-Codes sind nur druckende Zeichen, das heißt alle Zeichen außer den Steuerzeichen, vorgesehen. Die meisten Steuerzeichen führen zu Fehlfunktionen. Dennoch sind die Steuerzeichen CLEAR, HOME und CR (CR = CARRIAGE RETURN = CHR$(13) in Basic) zugelassen und wurden für die Window-Technik modifiziert. Mit HOME wird die linke obere Ecke des Windows angesprungen, mit CLEAR wird das Window gelöscht und automatisch HOME ausgeführt, und mit CR wird an den Anfang der nächsten Zeile gesprungen. Um Steuerzeichen wie REVERS ON/OFF oder Farbsteuerzeichen auszugeben, muß man diese über die normale BASOUT-Routine des Betriebssystems senden; die dadurch hervorgerufenen Effekte werden von der OUTW-Routine beibehalten, ein durch REVERS ON über BASOUT bewirkter Reversdruck gilt also auch für die von OUTW ausgegebenen Zeichen.

Von gleicher Einfachheit ist die Anwendung der PRWIN-Routine. In Akku (Low-Byte) und Y-Register (High-Byte) wird die Adresse übergeben, ab der der Text des Windows im Speicher steht. Am Ende des Textes und im Text sind keinerlei Endmarkierungen erforderlich, denn aus den Window-Grenzen geht eindeutig hervor, wieviele Zeichen ausgegeben werden müssen. Hier werden die Label LAENGE1 und LAENGE2 herangezogen.

Zwei Anwendungen von PRWIN-Routinen sehen Sie in den Zeilen 3150 bis 3300 und 3670 bis 3870.

Das mit Zeile 3880 beginnende Hauptprogramm ist nur eine Demonstration der Window-Technik. Das Grundlegende sind die Routinen zur Window-Technik. Wenn Sie diese in Ihren Quelltext übernehmen wollen, müssen Sie auch die Label LINKSn / RECHTSn / OBENn / UNTENn (n ist die Window-Nummer) ändern; die Berechnungsformel für LAENGEn (siehe Zeilen 590/600) muß auch in Ihrem Quelltext stehen, allerdings ist hier kein Umschreiben nötig.

Nun wünsche ich Ihnen noch viel Spaß mit den Window-Routinen und der Window-Technik im allgemeinen.

(Florian Müller/do)

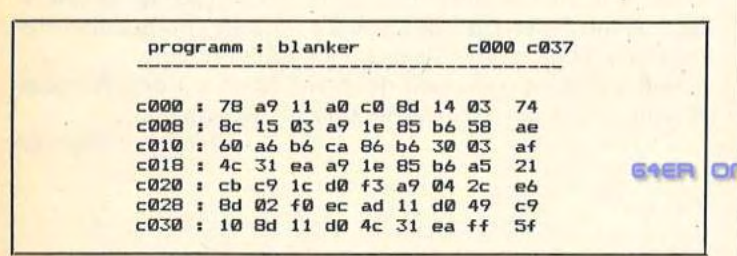


Der Blanker

Geschwindigkeit ist keine Hexerei. Immerhin läßt sich die Rechengeschwindigkeit mit dem »Blanker« um sagenhafte 5 Prozent steigern. Bei einer Stunde sind das ganze 3 Minuten. Während eines Compiler-Durchlaufs macht sich das bereits bemerkbar.

Das ganze Geheimnis dieses Programms ist, daß auf Tastendruck der Bildschirm abgeschaltet wird. Dadurch unterbricht der VIC den 6510 nicht mehr bei seinen Buszugriffen. Das Programm (siehe Listing 2 und Flußdiagramm) wird in den Interrupt eingebunden und prüft bei jedem dreißigsten Interrupt, ob die Tastenkombination <CTRL> <B> gedrückt ist. Ist dies der Fall, wird der Bildschirm ab- oder eingeschaltet. So lassen sich Compiler und langwierige Berechnungen in Basic beschleunigen. Geben Sie »Blanker« (Listing 1) bitte mit dem MSE ein und starten Sie ihn mit SYS 49152. Nach RUN/STOP RESTORE oder einem Reset ist ein Neustart erforderlich.

(M. Neetz/og)



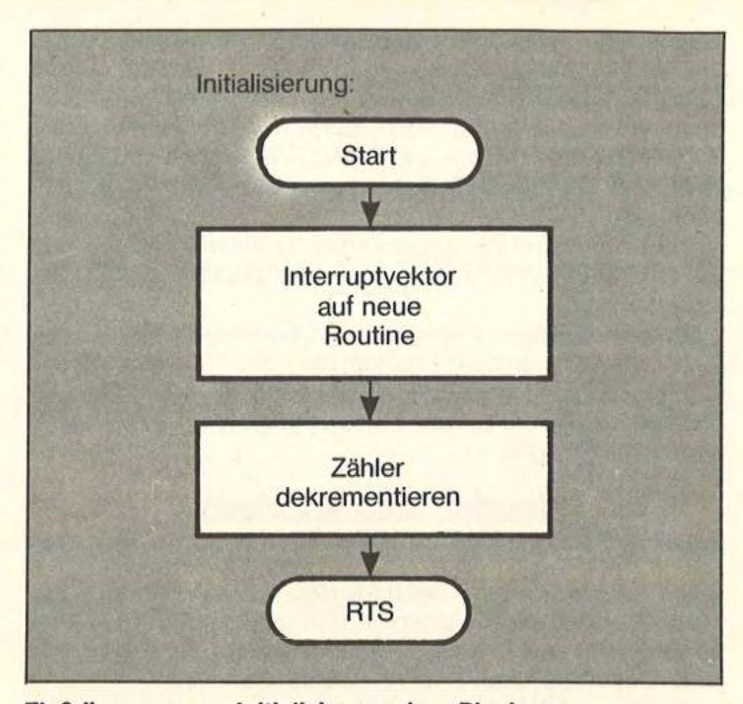
```
programm : blanker               c000 c037
------------------------------------------

c000 : 78 a9 11 a0 c0 8d 14 03   74
c008 : 8c 15 03 a9 1e 85 b6 58   ae
c010 : 60 a6 b6 ca 86 b6 30 03   af
c018 : 4c 31 ea a9 1e 85 b6 a5   21
c020 : cb c9 1c d0 f3 a9 04 2c   e6
c028 : 8d 02 f0 ec ad 11 d0 49   c9
c030 : 10 8d 11 d0 4c 31 ea ff   5f
```

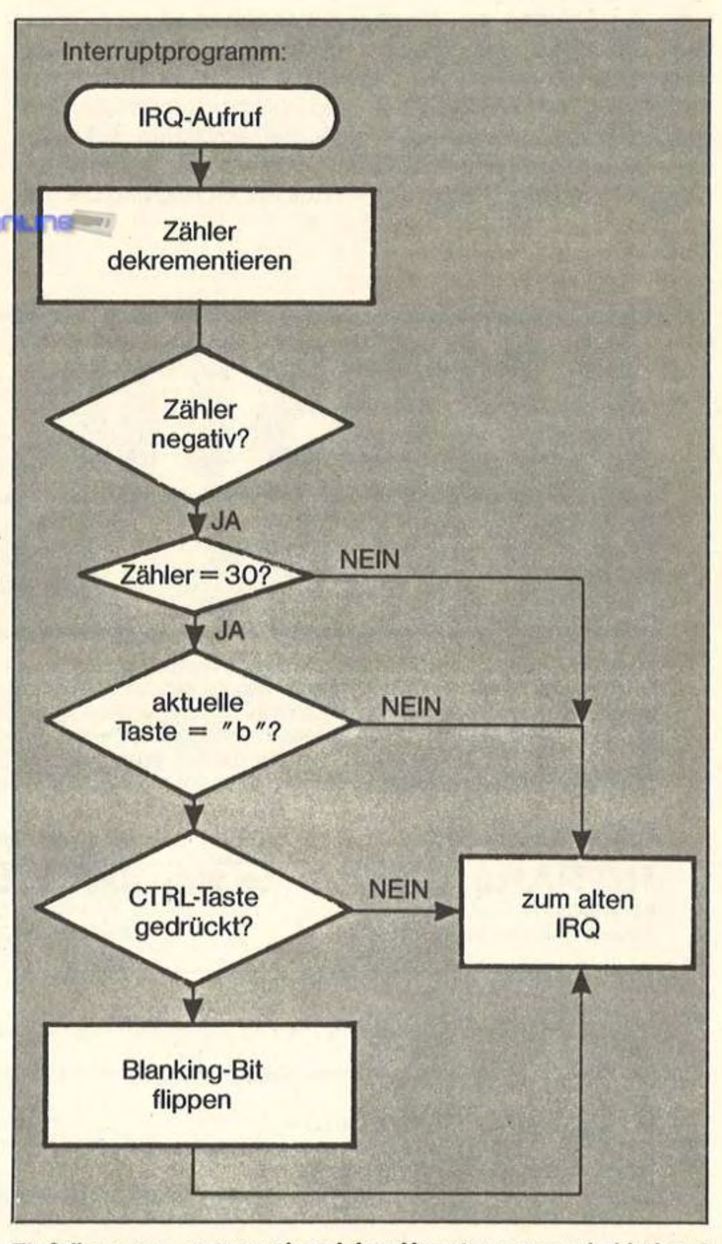
Listing 1. »Blanker« geben Sie mit dem MSE ein

```
100  -; ****** blanker ******
110  -; ** written 1986 by **
115  -; *** michael naatz ***
120  -;
130  -.eq irqvec   = $0314    ;irq-vektor
140  -.eq taste    = $cb      ;aktueller tastencode
150  -.eq altirq   = $ea31    ;adresse der alten irq-routine
160  -.eq blank    = $d011    ;vic-register 17
161  -.eq ctrlflag = $028d    ;flag fuer shift,c=,ctrl
165  -.eq zaehl    = $b6      ;zaehler
170  -;
180  -.ba $c000
185  -;
190  -; initialisierung
195  -;
200  -         sei            ;interrupts verhindern
210  -         lda #<(main)
220  -         ldy #>(main)
230  -         sta irqvec     ;irq-vektor setzen
240  -         sty irqvec+1
250  -         lda #30
260  -         sta zaehl      ;zaehler setzen
270  -         cli            ;interrupts zulassen
280  -         rts
290  -;
295  -; hauptprogramm
296  -;
300  -main     ldx zaehl
310  -         dex
320  -         stx zaehl      ;zaehler dekrementieren
330  -         bmi lab
340  -ende     jmp altirq     ;weiter beim alten irq
350  -lab      lda #30
360  -         sta zaehl      ;zaehler setzen
370  -         lda taste
380  -         cmp #28        ;auf "b" pruefen
390  -         bne ende
391  -         lda #04
392  -         bit ctrlflag   ;auf ctrl-taste pruefen
393  -         beq ende
400  -         lda blank      ;blank-bit flippen
410  -         eor #16
420  -         sta blank      ;und abspeichern
430  -         jmp altirq     ;weiter beim alten irq
```

Listing 2. Source-Code, mit Hypra-Ass erstellt zu »Blanker«



Flußdiagramm zur Initialisierung des »Blanker«



Flußdiagramm, entsprechend dem Hauptprogramm in Listing 2

Checksummer 64 V3

Der Checksummer 64 V3 überprüft jede Basic-Zeile direkt nach der Eingabe, erkennt Fehleingaben sowie Vertauschungen von Ziffern und erspart eine aufwendige Fehlersuche.

Der Checksummer 64 V3 ist ein kleines Maschinenprogramm, das Sie sofort unterrichtet, ob Sie die jeweilige Programmzeile korrekt eingegeben haben.

So gehen Sie vor:

1. Programm abtippen und speichern.

2. Starten mit RUN

3. Nach kurzer Zeit sehen Sie am Bildschirm: CHECKSUMMER 64, CHECKSUMMER AKTIVIERT, AUSSCHALTEN MIT POKE 1,55, ANSCHALTEN MIT POKE 1,53, READY.

4. Anschalten des Checksummer 64 V3 mit POKE 1,53.

5. Test: Geben Sie in einer freien Zeile ein: »1 REM« und drücken die RETURN-Taste. Am Bildschirm oben links sollten Sie die Prüfsumme <63> sehen.

6. Geben Sie ein Listing aus unserem Heft ein. Nach jeder Zeile wird die Zahl, die im Listing in Klammern < > steht, in den Bildschirm eingeblendet. Stimmen die Zahlen nicht überein, so liegt vermutlich ein Eingabefehler vor.

Die Zahl in den Klammern, und auch die Klammern selbst, dürfen beim Abtippen nicht mit eingegeben werden!

7. Der Checksummer 64 V3 bemerkt auch Vertauschungen von Zahlen und Buchstaben, aber nicht das Fehlen (oder Hinzufügen) von Leerzeichen.

8. Unsere Basic-Listings enthalten keine Steuerzeichen mehr. Diese werden ersetzt durch Klartext und stehen zwischen geschweiften Klammern. Deshalb sind weder die Klammern noch was dazwischen steht, abzutippen, sondern die in Tabelle 1 aufgeführten Tasten zu drücken. Auf Ihrem Bildschirm erhalten Sie dann wieder die entsprechenden Grafikzeichen.

9. Alle Grafikzeichen werden ebenfalls ersetzt durch unterstrichene oder überstrichene Großbuchstaben.

Unterstrichene Buchstaben bedeuten, daß Sie die SHIFT-Taste und den angegebenen Buchstaben drücken müssen, überstrichene jedoch die Commodore-Taste mit dem Buchstaben.

Auch hier erhalten Sie am Bildschirm das entsprechende Grafikzeichen und nicht etwa das im Listing erkennbare Zeichen.

Die Leerzeichen zwischen den einzelnen Basic-Befehlen können beim Abtippen entfallen (ohne Einfluß auf die Checksumme zu nehmen). Dies ist besonders bei speicherkritischen Programmen wichtig. Ebenso müssen Zeilen, die mehr als 80 Zeichen pro Zeile enthalten, mit den bekannten Abkürzungen für die Basic-Befehle (siehe auch das Handbuch zum C 64, Anhang D, Seite 130) eingegeben werden.

Sie können die Programme auch weiterhin ohne den Checksummer eintippen. (F. Lonczewski/gk)

Hinweis: {13 SPACE} bedeutet 13mal die Leertaste drücken

```
9 REM ********************************
10 PRINT"{CLR,11SPACE,RVSON}CHECKSUMMER 64
   V3{RVOFF}"
11 PRINT"{2DOWN,9SPACE}EINEN MOMENT, BITTE
   ..."
12 FOR I=828 TO 864:READ A:POKE I,A:PS=PS+
   A+1:NEXT I
13 IF PS<>5802 THEN PRINT"PRUEFSUMMENFEHLE
   R IN ZEILEN 20-22":END
14 SYS 828:PS=0:FOR I=58464 TO 58583:READ
   A:POKE I,A:PS=PS+A+1:NEXT I
15 IF PS<>16267 THEN PRINT"PRUEFSUMMENFEHL
   ER IN ZEILEN 22-30":END
16 POKE 1,53:POKE 42289,96:POKE 42290,228
17 PRINT"{4DOWN,9SPACE}CHECKSUMMER AKTIVIE
   RT."
18 PRINT"{2DOWN}AUSSCHALTEN : POKE1,55"
19 PRINT"{DOWN}ANSCHALTEN{2SPACE}: POKE1,5
   3":NEW
20 DATA 169,0,133,254,162,1,189,93,3,133,2
   55,160,0,177,254
21 DATA 145,254,136,208,249,230,255,165,25
   5,221,95,3,208,238,202
22 DATA 16,230,96,160,224,192,0,160,2,169,
   0,170,133,254,177
23 DATA 95,240,40,201,32,208,3,200,208,245
   ,133,255,138,41,7
24 DATA 170,240,14,72,165,255,24,42,105,0,
   202,208,249,133,255
25 DATA 104,170,232,165,255,24,101,254,133
   ,254,76,111,228,192,4
26 DATA 48,219,198,214,165,214,72,162,3,16
   9,32,157,1,4,189
27 DATA 212,228,32,210,255,208,12,0,92,72,
   32,201,255,170,104
28 DATA 144,1,138,96,202,16,228,166,254,16
   9,0,32,205,189,169
29 DATA 62,32,210,255,104,133,214,32,108,2
   29,169,141,32,210,255
30 DATA 76,128,164,9,60,18,19
```

© 64'er

Der Checksummer 64 V3 erkennt auch Vertauschungen von Zahlen

CTRL steht für Control-Taste, so bedeutet [CTRL-A], daß Sie die Control-Taste und die Taste »A« drücken müssen. Im folgenden steht:

Steuerbefehl	Taste
{DOWN}	Taste neben rechtem Shift, Cursor unten
{UP}	Shift-Taste & Taste neben rechtem Shift; Cursor hoch
{CLR}	Shift-Taste & 2. Taste ganz rechts oben
{INST}	Shift-Taste & Taste ganz rechts oben
{HOME}	2. Taste von ganz rechts oben
{DEL}	Taste ganz rechts oben
{RIGHT}	Taste ganz rechts unten
{LEFT}	Shift-Taste & Taste unten rechts
{SPACE}	Leertaste
{SHFT-SPCE}	Shift-Taste & Leertaste
{F1} bis {F8}	Funktionstasten
{RETURN}	Shift-Taste & Return
{BLACK}	Control-Taste & 1
{WHITE}	Control-Taste & 2
{RED}	Control-Taste & 3
{CYAN}	Control-Taste & 4
{PURPLE}	Control-Taste & 5
{GREEN}	Control-Taste & 6
{BLUE}	Control-Taste & 7
{YELLOW}	Control-Taste & 8
{RVSON}	Control-Taste & 9
{RVOFF}	Control-Taste & 0
{ORANGE}	Commodore-Taste & 1
{BROWN}	Commodore-Taste & 2
{LIG.RED}	Commodore-Taste & 3
{GREY 1}	Commodore-Taste & 4
{GREY 2}	Commodore-Taste & 5
{LIG.GREEN}	Commodore-Taste & 6
{LIG.BLUE}	Commodore-Taste & 7
{GREY 3}	Commodore-Taste & 8

Tabelle 1. Die Steuerbefehle in den Listings

MSE – Abtippen sicher und leicht gemacht

Ähnlich wie der »Checksummer« ist auch der MSE ein leicht zu bedienendes Hilfsmittel bei der Eingabe von Listings, diesmal jedoch bei reinen Maschinensprache-Programmen.

Im Gegensatz zum »Checksummer« aber ist die Eingabe nicht ohne den MSE möglich. Der MSE verringert die Tipparbeit um ein Drittel und schließt Fehleingaben vollkommen aus. Außerdem können Sie die Werte blind eingeben, ohne andauernd auf den Bildschirm schauen zu müssen. Dies wird durch akustische Meldungen realisiert.

MSE ist ein Maschinenspracheditor, mit dem ein Vertippen ausgeschlossen ist. Eine abgetippte Zeile wird nur angenommen, wenn sie richtig ist. Eine Checksumme am Ende jeder Zeile prüft, ob die richtigen Werte in der richtigen Zeile an der richtigen Stelle stehen. Wenn nicht, ertönt ein Warnsignal, und man beseitigt den Fehler.

War die Zeile korrekt, erklingt ein Gong, und die nächste Zeilennummer wird ausgegeben. Damit ist also auch »blindes« Eintippen möglich; Sie können sich voll auf den Text konzentrieren.

So arbeitet man mit MSE

Laden und starten Sie MSE. Zuerst wird der Programmname und die Start- und Endadresse erfragt. **Diese Angaben entnehmen Sie dem Kopf des jeweiligen abgedruckten Listings.** MSE meldet sich dann mit der Zeilennummer der ersten Zeile.

Wenn Sie die Zeile richtig eingegeben haben, erscheint die nächste Zeilennummer und so weiter bis zum Ende. Zum Schluß wird das fertige Programm mit »CTRL-S« auf Diskette oder Kassette abgespeichert. Dazu sind keine weiteren Angaben mehr erforderlich. Das Programm kann dann ganz normal wieder geladen und gestartet werden. Wenn Sie nicht alles auf einmal tippen wollen, können Sie jederzeit unterbrechen und den eingetippten Teil mit »CTRL-S« abspeichern. Wollen Sie weiterarbeiten, laden und starten Sie MSE wieder.

Geben Sie auf die Frage nach der Startadresse aber jetzt »L« ein, um Ihr Teilprogramm zu laden. Jetzt können Sie mit »CTRL-N« die Adresse eingeben, an der Sie weitertippen müssen. Wenn Sie sich nicht gemerkt haben, wie weit Sie gekommen sind, geben Sie nach dem Laden »CTRL-M« ein.

Auf die Frage nach der Startadresse antworten Sie mit der Anfangsadresse, die links in der Kopfzeile auf dem Bildschirm steht. Nun wird Ihr Programm aufgelistet. Mit »SPACE« wird das Listen fortgesetzt, mit »STOP« abgebrochen. Das Ende Ihres Programmteils erkennen Sie sehr einfach daran, daß nur noch der Wert »AA« in der Zeile steht. Die Adresse dieser Zeile müssen Sie anschließend mit »CTRL-N« eingeben. Das Programm ist nur mit »STOP/RESTORE« zu verlassen. Speichern Sie aber vorher unbedingt immer Ihren Text ab.

Hinweise zum Abtippen

Vor dem Abtippen oder späteren Wiederladen des MSE-**Laders** müssen Sie unbedingt folgende Zeile eingeben:
POKE 43,1: POKE 44,32: POKE 8192,0: NEW

Den MSE-**Lader** brauchen Sie nur einmal. Nach erfolgreichem Abtippen und Starten mit RUN geht der Lader verloren und es wird das endgültige Programm MSE V1.0 erzeugt. So gehen Sie vor:

Starten Sie das Programm mit RUN. Fehlerhafte Zeilen werden angezeigt und müssen korrigiert werden, bis der Lader zum »READY« durchläuft. Jetzt müssen Sie das fertige MSE-Programm speichern. Dazu brauchen Sie nur »RETURN« zu drücken, weil die erforderlichen Angaben schon auf dem Bildschirm stehen. (Kassettenbesitzer müssen in Zeile 343 die letzte Zahl in »1« abändern.) Ab jetzt können Sie »MSE V1.0« direkt, also ohne den DATA-Lader, benutzen. MSE V1.0 wird ganz normal mit »,8« geladen (keine POKEs notwendig).

(N. Mann/D. Weineck/gk)

MSE-Befehle:

DEL	löscht die letzte Eingabe.
CTRL-S	speichert das eingetippte Programm ab.
L oder CTRL-L	lädt ein Programm. Start- und Endadresse werden automatisch ermittelt.
CTRL-M	listet den Speicherinhalt. Abbruch mit STOP-Taste, weiter mit Leertaste.
CTRL-N	erlaubt die Eingabe einer neuen Adresse zum Weitertippen.
CTRL-P	gibt ein MSE-Listing auf dem Drucker aus.

```
100 REM ****************************           <091>
110 REM *                          *           <159>
120 REM *        M S E   LADER     *           <206>
130 REM *                          *           <179>
220 REM ****************************           <211>
230 REM                                        <036>
240 DIM H(75): FOR I=0 TO 9                    <113>
250 H(48+I)=I: H(65+I)=I+10:NEXT               <041>
260 FOR I=2048 TO 3755 : READ A$               <198>
270 H=ASC(LEFT$(A$,1)):L=ASC(RIGHT$(A$,1))     <199>
280 D=H(H)*16+H(L):S=S+D:POKE I,D              <219>
290 A=A+1:IF A<20 THEN NEXT:A=-1               <141>
300 PRINT " ZEILE:";1000+Z;                    <011>
310 READ V :Z=Z+1:IF V=S THEN 330              <218>
320 PRINT"PRUEFSUMMENFEHLER !":STOP            <138>
330 IF A<0 THEN 341                            <221>
340 S=0:A=0:PRINT:NEXT                         <046>
341 PRINT"{CLR}PO43,1:PO44,8:PO45,172:PO46
    ,14                                        <010>
342 POKE 631,19:POKE 632,13:POKE 633,13:PO
    KE 198,3                                   <249>
343 PRINT"{3DOWN}SAVE"CHR$(34)"MSE V1.0"CH
    R$(34)",8                                  <171>
344 END                                        <092>
1000 DATA 00,0B,08,0A,00,9E,32,30,36,31,00
     ,00,00,A2,08,A9,36,85,A4,A9, 1247         <119>
1001 DATA 08,85,A5,A9,00,85,A6,A9,B0,85,A7
     ,A0,00,B1,A4,91,A6,C8,D0,F9, 2888         <054>
1002 DATA E6,A5,E6,A7,CA,D0,F2,A9,36,85,01
     ,4C,00,B0,20,D1,B1,A9,06,8D, 2787         <144>
1003 DATA 21,D0,A9,03,8D,20,D0,8D,86,02,A0
     ,B3,A9,74,20,FF,B1,A0,B3,A9, 2667         <237>
1004 DATA B9,20,FF,B1,A0,00,20,CF,FF,99,01
     ,02,C8,C9,0D,D0,F5,88,F0,D2, 2912         <217>
1005 DATA C0,0F,90,02,A0,0E,8C,00,02,20,EA
     ,B1,A0,B3,A9,CF,20,FF,B1,20, 2323         <013>
1006 DATA 8E,B4,85,FC,85,62,20,8E,B4,85,FB
     ,85,61,20,A7,B4,D0,20,A0,B3, 2864         <199>
1007 DATA A9,E5,20,FF,B1,20,8E,B4,85,60,20
     ,8E,B4,85,5F,20,A7,B4,D0,0A, 2624         <091>
```

Der MSE zum bequemen Abtippen von Maschinenprogrammen

```
1008 DATA A5,61,C5,5F,A5,62,E5,60,90,06,20
     ,43,B3,4C,3A,B0,A9,AA,A0,00, 2379    <167>
1009 DATA 91,FB,E6,FB,D0,02,E6,FC,20,3F,B2
     ,90,EF,4C,FB,B4,A2,02,86,58, 3118    <152>
1010 DATA A9,A6,A0,9D,20,F2,B1,20,E4,FF,F0
     ,FB,C9,30,90,0C,C9,47,B0,08, 2970    <231>
1011 DATA C9,3A,90,0B,C9,41,B0,07,C9,14,D0
     ,0F,4C,0B,B1,20,D2,FF,A6,58, 2322    <121>
1012 DATA 95,F7,C6,58,D0,D2,60,AE,8D,02,F0
     ,26,C9,0C,D0,03,4C,0B,B6,C9, 2685    <057>
1013 DATA 13,D0,03,4C,8B,B5,C9,0D,D0,03,4C
     ,BA,B4,C9,10,D0,03,4C,68,B5, 2282    <225>
1014 DATA C9,0E,D0,06,20,5F,B4,4C,64,B1,4C
     ,92,B0,A5,F9,20,02,B1,0A,0A, 2132    <208>
1015 DATA 0A,0A,85,F9,A5,F8,20,02,B1,05,F9
     ,60,C9,3A,90,02,69,08,29,0F, 1950    <092>
1016 DATA 60,A6,59,E0,08,90,1F,A6,58,E0,02
     ,B0,06,20,D2,FF,4C,8E,B0,C6, 2509    <188>
1017 DATA 59,A0,14,A9,92,20,F2,B1,CA,D0,FA
     ,84,57,68,68,4C,8B,B1,A6,D3, 2891    <197>
1018 DATA E0,08,B0,03,4C,92,B0,20,D2,FF,A6
     ,58,E0,02,90,09,C6,59,20,D2, 2468    <049>
1019 DATA FF,C6,58,D0,F9,4C,8E,B0,48,4A,4A
     ,4A,4A,20,59,B1,68,29,0F,C9, 2419    <035>
1020 DATA 0A,90,02,69,06,69,30,4C,D2,FF,A2
     ,FC,9A,20,D1,B1,20,48,B2,20, 2261    <073>
1021 DATA EA,B1,20,9F,B2,A5,FC,20,4E,B1,A5
     ,FB,20,4E,B1,20,ED,B1,A9,3A, 2860    <148>
1022 DATA A0,20,20,F2,B1,A9,00,85,59,20,8E
     ,B0,20,ED,B1,A4,59,20,EF,B0, 2530    <233>
1023 DATA 91,FB,C8,84,59,C0,08,90,EC,20,10
     ,B2,A9,12,20,D2,FF,20,8E,B0, 2657    <105>
1024 DATA 20,EF,B0,C5,FF,F0,0D,20,43,B3,A9
     ,14,A0,14,20,F2,B1,4C,A2,B1, 2665    <034>
1025 DATA A9,92,20,D2,FF,20,33,B2,20,E0,B2
     ,20,3F,B2,90,9F,4C,8B,B5,A9, 2648    <123>
1026 DATA 93,20,D2,FF,A2,00,A9,03,9D,00,D8
     ,9D,00,D9,9D,00,DA,9D,00,DB, 2476    <237>
1027 DATA E8,D0,EF,60,A9,0D,2C,A9,20,4C,D2
     ,FF,20,D2,FF,98,4C,D2,FF,20, 2965    <160>
1028 DATA E4,FF,F0,FB,60,84,5D,85,5C,A0,00
     ,B1,5C,F0,06,20,D2,FF,C8,D0, 3100    <077>
1029 DATA F6,60,A5,FB,85,5A,A0,00,84,5B,B1
     ,FB,18,65,5A,85,5A,90,02,E6, 2606    <156>
1030 DATA 5B,06,5A,26,5B,C8,C0,08,90,EC,A5
     ,5A,65,5B,85,FF,60,18,A5,FB, 2467    <219>
1031 DATA 69,08,85,FB,90,02,E6,FC,60,A5,FB
     ,C5,5F,A5,FC,E5,60,60,A0,B3, 3106    <183>
1032 DATA A9,FB,20,FF,B1,A0,01,B9,00,02,20
     ,D2,FF,CC,00,02,C8,90,F4,A9, 2692    <098>
1033 DATA 10,ED,00,02,AA,20,ED,B1,CA,D0,FA
     ,A5,62,20,4E,B1,A5,61,20,4E, 2453    <236>
1034 DATA B1,20,ED,B1,A5,60,20,4E,B1,A5,5F
     ,20,4E,B1,A9,9F,20,D2,FF,20, 2575    <038>
1035 DATA EA,B1,24,5E,10,01,60,A9,12,20,D2
     ,FF,A2,28,20,ED,B1,CA,D0,FA, 2646    <161>
1036 DATA A9,92,4C,D2,FF,A5,D6,C9,16,B0,01
     ,60,A9,A0,85,A4,A9,78,85,A6, 2945    <204>
1037 DATA A9,04,85,A5,85,A7,A2,13,A0,27,B1
     ,A4,91,A6,88,10,F9,CA,F0,19, 2671    <208>
1038 DATA 18,A5,A4,69,28,85,A4,90,02,E6,A5
     ,18,A5,A6,69,28,85,A6,90,E0, 2503    <251>
1039 DATA E6,A7,4C,B6,B2,A9,91,4C,D2,FF,A9
     ,0F,8D,18,D4,A9,00,8D,05,D4, 2776    <000>
1040 DATA A9,F7,8D,06,D4,A9,11,8D,04,D4,A9
     ,32,8D,01,D4,A9,00,8D,00,D4, 2413    <126>
1041 DATA A0,80,20,09,B3,A9,10,8D,04,D4,60
     ,A2,FF,CA,D0,FD,88,D0,F8,60, 2914    <240>
1042 DATA A9,0F,8D,18,D4,A9,2D,8D,05,D4,A9
     ,A5,8D,06,D4,A9,21,8D,04,D4, 2385    <119>
1043 DATA A9,07,8D,01,D4,A9,05,8D,00,D4,A0
     ,FF,20,09,B3,A9,20,8D,04,D4, 2250    <078>
1044 DATA A9,00,8D,01,D4,8D,00,D4,60,38,20
     ,F0,FF,8A,48,98,48,18,A0,06, 2179    <175>
1045 DATA A2,18,20,F0,FF,A0,B4,A9,0A,20,FF
     ,B1,20,12,B3,20,E4,FF,F0,FB, 2931    <093>
1046 DATA A2,1D,A9,14,20,D2,FF,CA,D0,FA,68
     ,A8,68,AA,18,4C,F0,FF,0D,0D, 2704    <088>
1047 DATA 0D,20,20,20,20,20,20,20,4D,41,53
     ,43,48,49,4E,45,4E,53,50,52, 1144    <216>
1048 DATA 41,43,48,45,20,2D,20,45,44,49,54
     ,4F,52,20,0D,0D,20,20,20,20, 1023    <038>
1049 DATA 20,20,20,20,56,4F,4E,20,4E,2E,4D
     ,41,4E,4E,20,26,20,44,2E,57, 1128    <206>
1050 DATA 45,49,4E,45,43,4B,00,0D,0D,0D,20
     ,20,20,50,52,4F,47,52,41,4D, 1102    <117>
1051 DATA 4D,4E,41,4D,45,20,3A,20,00,0D,0D
     ,20,20,20,53,54,41,52,54,41, 1073    <095>
1052 DATA 44,52,45,53,53,45,20,3A,20,24,00
     ,0D,0D,20,20,20,45,4E,44,41, 1014    <129>
1053 DATA 44,52,45,53,53,45,20,20,20,3A,20
     ,24,00,92,05,20,50,52,4F,47, 1171    <217>
1054 DATA 52,41,4D,4D,20,3A,20,00,12,20,20
     ,2A,2A,2A,20,46,41,4C,53,43, 1024    <027>
1055 DATA 48,45,20,45,49,4E,47,41,42,45,20
     ,2A,2A,2A,20,20,92,00,0D,0D, 1058    <098>
1056 DATA 2A,2A,2A,20,45,4E,44,45,20,2A,2A
     ,2A,00,13,05,20,20,12,44,92, 920     <148>
1057 DATA 49,53,4B,20,4F,44,45,52,20,12,54
     ,92,41,50,45,0D,00,13,20,20, 1151    <035>
1058 DATA 49,2F,4F,20,2D,20,46,45,48,4C,45
     ,52,00,20,D1,B1,20,48,B2,A0, 1606    <012>
1059 DATA B3,A9,CF,20,FF,B1,20,8E,B4,85,FC
     ,20,8E,B4,85,FB,C5,61,A5,FC, 3207    <251>
1060 DATA E5,62,90,23,A5,FB,C5,5F,A5,FC,E5
     ,60,B0,19,20,A7,B4,D0,14,60, 2860    <112>
1061 DATA 20,A7,B4,F0,0C,85,F9,20,A7,B4,F0
     ,05,85,F8,4C,EF,B0,68,68,20, 2749    <088>
1062 DATA 43,B3,4C,5F,B4,20,CF,FF,C9,4C,D0
     ,09,20,D1,B1,20,48,B2,4C,0B, 2372    <046>
1063 DATA B6,C9,0D,60,A9,00,85,5E,20,5F,B4
     ,20,EA,B1,20,0D,B5,24,5E,30, 2042    <120>
1064 DATA 05,20,E4,FF,F0,FB,20,E1,FF,F0,26
     ,20,9F,B2,24,5E,10,09,20,4E, 2435    <198>
1065 DATA B5,20,0D,B5,20,60,B5,20,33,B2,20
     ,3F,B2,90,D7,A0,B4,A9,28,20, 2190    <207>
1066 DATA FF,B1,20,E4,FF,C9,0D,D0,F9,A9,00
     ,85,5E,A5,61,85,FB,A5,62,85, 3056    <240>
1067 DATA FC,20,E0,B2,4C,64,B1,A5,FC,20,4E
     ,B1,A5,FB,85,FF,20,4E,B1,A9, 3003    <221>
1068 DATA 20,A0,3A,20,F2,B1,A0,00,20,ED,B1
     ,B1,FB,20,4E,B1,C8,C0,08,90, 2566    <070>
1069 DATA F3,20,ED,B1,24,5E,30,03,A9,12,2C
     ,A9,20,20,D2,FF,20,10,B2,A5, 2190    <059>
1070 DATA FF,20,4E,B1,A9,92,20,D2,FF,4C,EA
     ,B1,A9,FF,85,B8,85,B9,A9,04, 3073    <029>
1071 DATA 85,BA,20,C0,FF,A2,FF,4C,C9,FF,20
     ,CC,FF,A9,FF,4C,C3,FF,20,5F, 3315    <189>
1072 DATA B4,A9,80,85,5E,20,4E,B5,20,48,B2
     ,A2,24,A9,2D,20,D2,FF,CA,D0, 2596    <111>
1073 DATA FA,20,EA,B1,20,EA,B1,20,60,B5,4C
     ,C1,B4,20,B8,B5,A6,5F,A4,60, 2812    <015>
1074 DATA A9,61,20,D8,FF,B0,0A,20,B7,FF,29
     ,BF,D0,03,4C,FB,B4,A9,01,20, 2577    <201>
1075 DATA C3,FF,20,68,B6,A0,B4,A9,4F,20,FF
     ,B1,20,F9,B1,4C,FB,B4,20,68, 2921    <237>
1076 DATA B6,A9,37,A0,B4,20,FF,B1,20,F9,B1
     ,A2,08,C9,44,F0,06,A2,01,C9, 2717    <213>
1077 DATA 54,D0,F1,A9,01,A8,20,BA,FF,A0,00
     ,E0,01,F0,1A,A9,40,8D,20,02, 2403    <101>
1078 DATA A9,3A,8D,21,02,B9,01,02,99,22,02
     ,C8,CC,00,02,90,F4,C8,C8,D0, 2182    <127>
1079 DATA 0C,B9,01,02,99,20,02,C8,CC,00,02
     ,D0,F4,98,A2,20,A0,02,4C,BD, 2018    <025>
1080 DATA FF,20,B8,B5,A5,BA,C9,08,90,33,A6
     ,B9,86,57,A9,01,20,C3,FF,A9, 2800    <022>
1081 DATA 60,85,B9,20,C0,FF,B0,28,A5,BA,20
     ,B4,FF,A5,B9,20,96,FF,20,A5, 2911    <053>
1082 DATA FF,85,61,A5,90,4A,4A,B0,13,20,A5
     ,FF,85,62,20,AB,FF,A5,57,85, 2663    <214>
1083 DATA B9,A9,00,20,D5,FF,90,03,4C,A3,B5
     ,86,5F,84,60,A5,BA,C9,01,D0, 2639    <131>
1084 DATA 0A,AD,3D,03,85,61,AD,3E,03,85,62
     ,4C,FB,B4,A9,13,20,D2,FF,A2, 2300    <120>
1085 DATA 1C,20,ED,B1,CA,D0,FA,60, 1230    <214>
```


MSE (Schluß). Dieses Listing können Sie (müssen aber nicht) mit dem Checksummer 64 V3 in diesem Heft eingeben.

Checksummer für C 128

Auch für den C 128 gibt es jetzt eine Eingabehilfe. Der Checksummer 128 erkennt Fehler, die Sie beim Eintippen gemacht haben. Dadurch ersparen Sie sich eine aufwendige Fehlersuche.

Der Checksummer 128 ist ein Maschinenprogramm, das Sie nach jeder eingegebenen Basic-Zeile unterrichtet, ob bei der Eingabe Fehler gemacht wurden oder nicht.

Zu diesem Zweck steht rechts neben jeder Basic-Zeile eine Prüfsumme in eckigen Klammern. Beim Abtippen von C 128-Listings dürfen Sie weder die Klammern noch die Prüfsumme mit eingeben.

Der Checksummer 128 ermittelt, nachdem Sie eine Basic-Zeile mit <RETURN> eingegeben haben, eine Prüfsumme und zeigt sie am Bildschirm in der oberen linken Ecke an. Wenn diese Zahl mit der Prüfsumme neben dem Listing übereinstimmt, ist alles in Ordnung. Falls nicht, haben Sie einen Fehler gemacht und können ihn sofort verbessern.

Eingabehinweise

So gehen Sie vor:

1. Tippen Sie das Listing »Lader.Checksummer 128« ab.

2. Speichern auf Diskette oder Kassette.

3. Starten mit RUN.

4. Am Bildschirm werden die Zeilennummern ab 1000 der Reihe nach geLISTet. Ein Fehler beim Abtippen wird mit großer Wahrscheinlichkeit erkannt. Wenn die Meldung »ZEILE xxxx PRUEFSUMMENFEHLER ! xxxx« erscheint, haben Sie in der Zeile xxxx einen Fehler gemacht, den Sie sofort korrigieren können. Danach erneut mit RUN starten.

5. Wenn der Lader.Checksummer 128 fehlerfrei durchgelaufen ist, kann er mit SYS 2820 gestartet werden. Eine Meldung wird nicht ausgegeben. Das Listing wird dabei gelöscht.

6. Geben Sie zur Kontrolle ein: 2 REM und drücken <RETURN>. Am Bildschirm sollte oben links die Prüfsumme <68P> erscheinen.

7. Im Speicher steht jetzt der fertige Checksummer 128 als Maschinenprogramm. Damit Sie in Zukunft nicht mehr den Lader benutzen brauchen, speichern Sie sich das Maschinenprogramm auf Diskette oder Kassette. Gehen Sie so vor:

8. Rufen Sie mit <F8> den Monitor auf. Geben Sie dann ein:

```
S"checksum128.2820",08 0b22 0d64
```

Bei Kassettenbetrieb muß anstatt »,08« »,01« eingegeben werden. Der Checksummer 128 wird jetzt unter dem Namen »checksum128.2820« gespeichert und kann mit demselben Namen und ,8,1 geladen werden. Die 2820 im Namen zeigt Ihnen auch in Zukunft, daß das Programm mit SYS 2820 gestartet wird.

So geben Sie die Listings ein

In unseren Listings finden Sie keine Steuer- und Grafikzeichen mehr. Sie wurden ersetzt durch Klartext. Aus dem reversen Herzen für Bildschirm löschen wird nun zum Beispiel {CLR}. Das heißt, Sie dürfen nicht die Klammern eingeben und die Buchstaben CLR, sondern Sie sollen die Taste <CLR> drücken. Weitere Informationen finden Sie im Artikel »Checksummer 64 V3«.

Hinweis für C 64-Besitzer

Auch als »nur« C 64-Besitzer können Sie diesen Checksummer benutzen. Er ist so ausgelegt, daß er auf beiden Computern sowohl im C 128- als auch im C 64-Modus läuft.

```
1 REM ***********************************
2 REM *    ++ CHECKSUMMER 128/64++       *
3 REM *          GERD MOELLMANN          *
7 REM ***********************************
8 :
9 :
10 DIM H(75) : FOR I=0 TO 9
20 H(48+I)=I : H(65+I)=I+10 : NEXT
30 FOR I=2850 TO 3428 : READ A$
40 H=ASC(LEFT$(A$,1)):L=ASC(RIGHT$(A$,1))
50 D=H(H)*16+H(L) : S=S+D*(A+1): POKE I,D

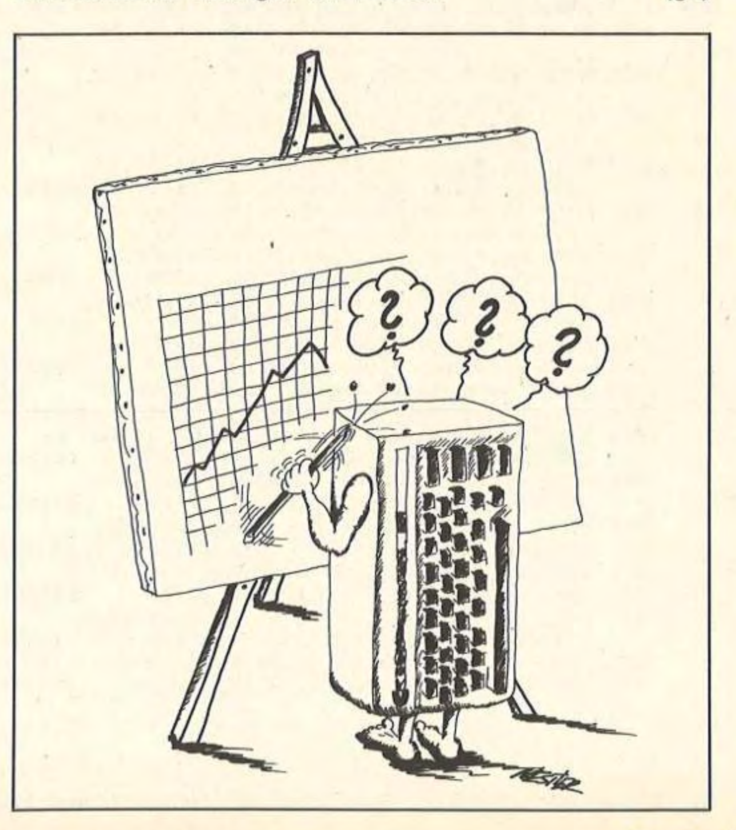
60 A=A+1:IF A<9  THEN NEXT : A=-1
65 PRINT "ZEILE:";1000+Z;
70 READ V : Z=Z+1 : IF V=S THEN 85
80 PRINT" PRUEFSUMMENFEHLER !";999+Z:STOP
85 IF A<0 THEN END
90 S=0 : A=0 : PRINT : NEXT : END
95 :
1000 DATA 4C,2A,0B,00,00,00,00,00,A9, 1714
1001 DATA 00,8D,00,FF,A9,0D,20,D2,FF, 6424
1002 DATA AD,CF,41,C9,56,D0,11,AD,D0, 6639
1003 DATA 41,C9,37,D0,0A,AD,D1,41,C9, 6344
1004 DATA 2E,D0,03,A9,FF,2C,A9,00,8D, 5138
1005 DATA 25,0B,AA,D0,0B,8D,64,0D,A9, 4627
1006 DATA 65,85,2B,A9,0D,85,2C,20,CD, 4444
1007 DATA 0B.A9,6C,8D,02,03,A9,0B,8D, 3805
1008 DATA 03,03,2C,27,0B,10,06,EE,27, 2745
```

Bevor Sie den Lader.Checksummer 128 abtippen oder laden, müssen Sie folgende Zeile eingeben:

POKE 43,0:POKE 44,16:POKE 4095,0:NEW <RETURN>

Anschließend verfahren Sie genauso wie oben schon beschrieben. Da Sie jedoch keinen eingebauten Monitor besitzen, müssen Sie einen anderen nehmen, zum Beispiel den SMON.

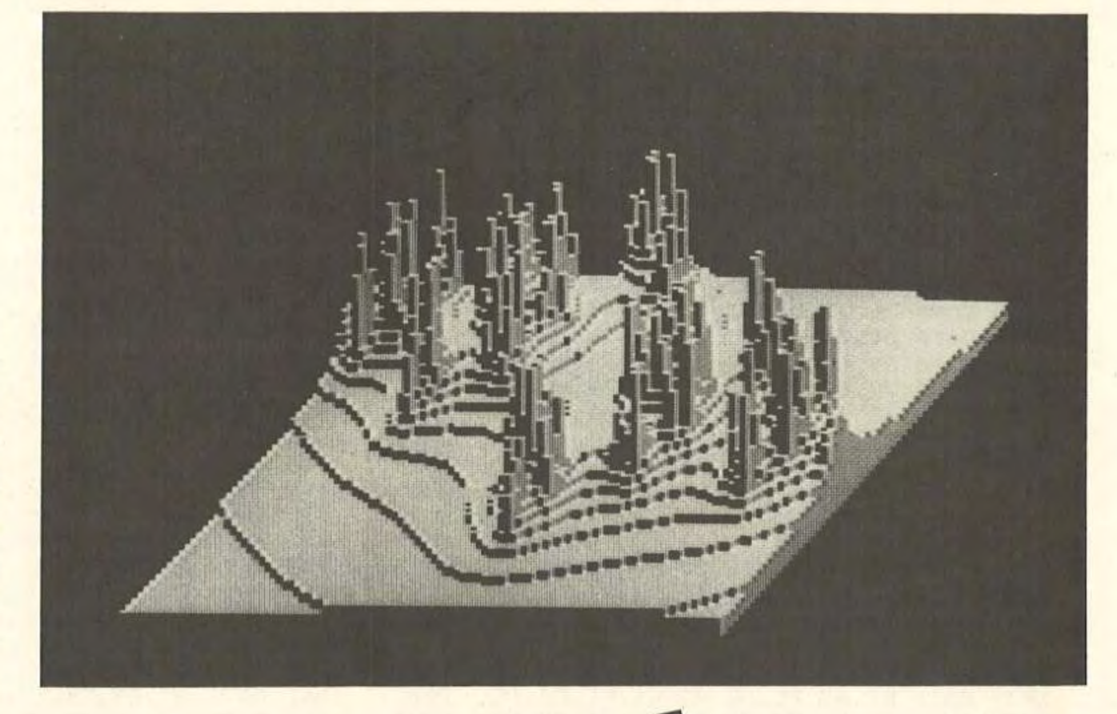
Der Checksummer 128 wird in Zukunft den Checksummer 64 V3 ablösen, da er erstens für beide Computer geschrieben wurde, zweitens besser ist und last not least, weil zwei Checksummer-Listings Platz kosten. (gk)



```
1009 DATA 0B,4C,35,0D,2C,26,0B,10,03, 1054
1010 DATA 4C,F9,0C,20,A4,0B,20,AF,0B, 3347
1011 DATA 20,C2,0B,AA,F0,E1,B0,0C,CE, 6865
1012 DATA 26,0B,20,E3,0B,20,AF,0B,20, 2912
1013 DATA C2,0B,2C,25,0B,30,03,4C,94, 2800
1014 DATA A4,4C,D7,4D,2C,25,0B,30,03, 2199
1015 DATA 4C,60,A5,4C,93,4F,A9,FF,A2, 6957
1016 DATA 01,2C,25,0B,30,05,85,7A,86, 3627
1017 DATA 7B,60,85,3D,86,3E,60,2C,25, 3357
1018 DATA 0B,30,03,4C,73,00,4C,80,03, 2578
1019 DATA 2C,25,0B,30,03,4C,44,A6,4C, 3302
1020 DATA D9,51,2C,25,0B,30,03,4C,13, 1802
1021 DATA A6,4C,64,50,2C,25,0B,30,06, 1895
1022 DATA 20,6B,A9,4C,F1,0B,20,A0,50, 4552
1023 DATA A2,00,2C,25,0B,10,02,A2,02, 1921
1024 DATA B5,15,85,AA,B5,14,85,A9,60, 5474
1025 DATA A9,00,85,A7,85,A8,A9,FF,85, 7329
1026 DATA FC,A9,07,85,FD,20,9D,0C,F0, 5955
1027 DATA FB,C9,30,90,17,C9,3A,B0,13, 4679
1028 DATA 20,8E,0C,4C,11,0C,20,9D,0C, 2401
1029 DATA D0,08,20,9D,0C,F0,FB,A2,01, 5510
1030 DATA 2C,A2,00,C9,52,D0,26,20,9D, 4765
1031 DATA 0C,C9,45,D0,17,20,9D,0C,C9, 4764
1032 DATA 4D,D0,08,20,9D,0C,C9,3A,D0, 5245
1033 DATA F9,60,A5,FC,D0,02,C6,FD,C6, 8188
1034 DATA FC.A5.FC.D0.02,C6.FD.C6.FC. 8991
1035 DATA A5,FC,D0,02,C6,FD,C6,FC,8A, 8453
1036 DATA F0,05,A9,3A,20,8E,0C,20,9D, 3754
1037 DATA 0C,C9,20,F0,F9,C9,3A,F0,B1, 7840
1038 DATA C9,22,F0,06,20,8E,0C,4C,6D, 3698
1039 DATA 0C,20,8E,0C,20,9D,0C,C9,22, 3650
1040 DATA D0,F6,F0,EE,06,A7,26,A8,08, 5086
1041 DATA 46,A7,28,26,A7,45,A7,85,A7, 5661
1042 DATA 60,E6,FC,D0,02,E6,FD,A0,00, 6585
1043 DATA B1,FC,F0,07,C9,20,F0,02,C9, 6131
1044 DATA 3A,60,68,68,60,A5,A8,45,A7, 5679
1045 DATA 85,A8,20,BF,0C,20,BF.0C.A9. 4535
1046 DATA 00,A2,04,06,A7,26,A8,2A,CA, 4753
1047 DATA 10,F8,C9,0A,90,02,69,06,69, 3615
1048 DATA 30,4C,D2,FF,A9,00,8D,00,FF, 5977
1049 DATA 38,20,F0,FF,8A,48,A9,13,20, 4605
1050 DATA D2,FF,A9,3C,20,D2,FF,20,B3, 6539
1051 DATA 0C,A9,3E,20,D2,FF,68,AA,18, 5548
1052 DATA 4C,F0,FF,AD,27,03,48,AD,26, 4456
1053 DATA 03,48,A9,49,8D,26,03,A9,0D, 3369
1054 DATA 8D,27,03,2C,25,0B,10,03,A2, 2249
1055 DATA 02,2C,A2,00,A5,A9,95,14,A5, 5103
1056 DATA AA,95,15,20,D8,0B,A9,00,8D, 4257
1057 DATA 28,0B,2C,25,0B,30,08,CE,27, 2740
1058 DATA 0B,A0,02,4C,D8,A6,20,F8,50, 5645
1059 DATA 20,03,0C,EE,26,0B,68,8D,26, 3480
1060 DATA 03,68,8D,27,03,20,D6,0C,4C, 3275
1061 DATA 6C,0B,85,AB,8A,48,AE,28,0B, 3972
1062 DATA A5,AB,C9,0D,D0,02,A9,00,9D, 4810
1063 DATA 00,08,EE,28,0B,68,AA,A5,AB, 5618
1064 DATA 18,60,41, 411
```

Listing. »Lader.Checksummer 128« für den Commodore 128

Apfelmännchen sind nicht nur als Ebene darstellbar. Zeichnen Sie sie mit dreidimensionalen Effekten und in räumlicher Tiefe. Die Grafiken sehen aus wie Hochhaus- oder Gebirgs-Landschaften.





Vergleichen Sie das nebenstehende Bild mit einem Apfelmännchen aus Ausgabe 11/85, wird Ihnen die dem Programm zugrunde liegende Idee sofort auffallen. Die Anzahl der Iterationen bestimmt nun nicht mehr einen Farbwert, vielmehr stellt sie die Höhe eines Punktes über der Ebene dar. Die Höhe wird als Balken gezeichnet. Dabei besteht jeder Balken aus einer dunklen Vorderseite, einer etwas helleren rechten Seite und einer hellen Oberseite. Da diese Balken von links nach rechts und von hinten nach vorne aneinandergefügt werden, benötigt das Programm keinen Algorithmus zur Berechnung verdeckter Linien.

Das Programm »FRACTALBERGE« (siehe Listing 1) ist für Simons Basic geschrieben. Benötigt werden jedoch nur die Befehle zum Einschalten der Grafik und zum Zeichnen der Linien. Die das Bild bestimmenden Parameter sind in den Zeilen 5 bis 7 enthalten (Listing 2) und gliedern sich wie folgt.

Zeile 5: Festlegen der Farben, Gesamtgröße des Bildes
Zeile 6: Komplexe Parameter, Maximale Anzahl der Iterationen, maximale Höhe der Balken
Zeile 7: Ausschnitt des Bildes festlegen

Es wird immer ein rechteckförmiger Ausschnitt der Fractalebene dargestellt (Bild 1). Die Parallel-Perspektive entsteht durch das Verschieben jeder Zeile gegenüber der vorhergehenden um einen halben Bildpunkt nach links. Die Laufzeit beträgt im Durchschnitt über eine Stunde und ist je nach Bild nach oben beliebig offen. (G.Paret/og)

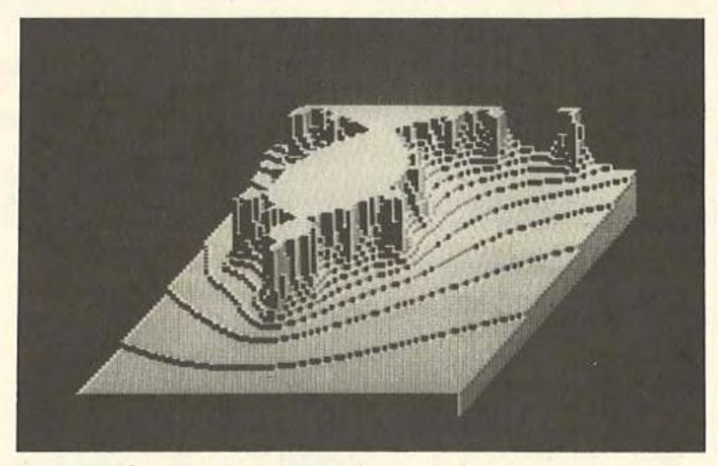


Bild 1. Apfelmännchen, fast wie ein Hochplateau

```
5 HIRES1,0:COLOUR6,6:MULTI7,10,0:XM=105:YM=105
6 XC=1:YC=0:T=20:S=60
7 XL=-.15:XR=.26:YO=.47:YU=.9
8 DX=(XR-XL)/XM:DY=(YU-YO)/YM
9 FORN=0TOYM:Y1=YO+N*DY:FORM=0TOXM:X=XL+M*DX:Y=Y1:
K=0
10 X2=X*X:Y2=Y*Y:Y=2*X*Y-YC:X=X2-Y2-XC:K=K+1:IF(K<
T)AND(X2+Y2<S)THEN10
11 U=M+53-N/2:U1=U+1:V=N+80:V1=V-3*(K-1)
12 LINE U,V,U,V1,3:LINE U1,V,U1,V1,2:LINE U,V1,U1,
V1,1
13 NEXT:NEXT
14 WAIT 198,255
```

Listing 1. »FRACTALBERGE« ist in Simons Basic geschrieben. Die Parameter entsprechen dem oberen Bild

```
5 HIRES 1,0:COLOUR 0,0:MULTI 7,10,9:XM=105:YM=105
6 XC=.77:YC=.6:T=22:S=40
7 XL=-1.4:XR=1.4:YO=-.9:YU=1.2
8 DX=(XR-XL)/XM:DY=(YU-YO)/YM
9 FORN=0TOYM:Y1=YO+N*DY:FORM=0TOXM:X=XL+M*DX:Y=Y1:
K=0
10 X2=X*X:Y2=Y*Y:Y=2*X*Y-YC:X=X2-Y2-XC:K=K+1:IF(K<
T)AND(X2+Y2<S)THEN10
11 U=M+53-N/2:U1=U+1:V=N+80:V1=V-3*(K-1)
12 LINE U,V,U,V1,3:LINE U1,V,U1,V1,2:LINE U,V1,U1,
V1,1
13 NEXT:NEXT
14 WAIT 198,255
```

Listing 2. Ein Beispiel für andere Parameter, entsprechend Bild 1



Basic-Programme kürzen

Mit Hilfe dieses kleinen Programms können Sie Speicherplatz bei Basic-Programmen sparen. Es entfernt alle überflüssigen Bytes aus Ihrem Programm.

Dieses Programm verkürzt jedes Basic-Programm, indem nicht benötigte Leerzeichen, REM-Zeilen und so weiter aus dem Programm entfernt werden. Zusätzlich können als Option die Basic-Zeilen, die Editor-bedingt nur 88 Zeichen lang sein können, auf bis zu 255 Zeichen pro Zeile zusammengeschoben werden. Dies ist gelegentlich notwendig, wenn Sie zum Beispiel ein Adreß- oder Dateiverwaltungsprogramm geschrieben haben und dieses zur besseren Übersichtlichkeit gut mit REM-Zeilen dokumentierten. Im nachhinein stellen Sie aber fest, daß Ihnen Ihr Speicher für die Variablen nicht mehr ganz ausreicht (OUT OF MEMORY ERROR) und Sie dringend Platz benötigen.

In diesem Falle hilft es Ihnen, das Hauptprogramm zu verkürzen, indem Sie alle nicht benötigten Bytes entfernen. Dies sind REM-Zeilen, nicht benötigte Spaces (Leerzeichen) oder ähnliches. Ebenfalls hilft es, die Basic-Zeilen neu zu »linken«, also mehrere Zeilen zu einer einzigen neu zu binden. Hierbei entfallen pro gesparter Zeilennummer 4 Byte für die Zeilennummer und den Linkpointer auf die nächste Zeile.

Dadurch kann oft sehr viel Speicherplatz gespart werden.

Danach ist das Programm zwar nicht mehr editierbar und die Lesbarkeit des Programms hat auch etwas gelitten, aber es läuft genauso (meist sogar etwas schneller) wie das Original (bedingt dadurch, daß der Interpreter jetzt nicht mehr soviel Programm »durchforsten« muß), und außerdem hat man noch Speicherplatz gespart.

Und das ist bei vielen Basic-Programmen ja auch der Effekt, den man erreichen will.

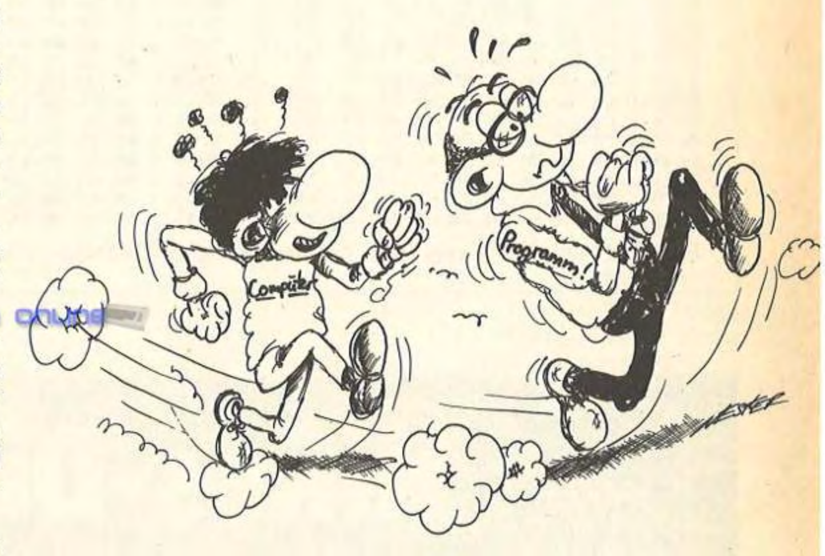
Eingabehinweise

Das Programm (Listing 1) geben Sie bitte mit Hilfe des MSE ein und speichern es auf Ihren Datenträger.

Danach läßt sich der Basic-Packer normal mit »LOAD "BASIC-PACKER",8« laden und durch RUN starten. Er verschiebt sich selbständig in den $C000-Bereich (Aufruf mit SYS 49152).

Nach Abschluß des Packens wird Ihnen noch angezeigt, wieviele Bytes eingespart wurden. Jetzt ist das Programm gepackt und bereit zum Speichern.

(Christoph Zwerschke/dm)




```
name : basic-packer      0801 0f48
--------------------------------------
0801 : 23 08 00 00 9e 32 30 38  d5
0809 : 38 14 14 14 14 14 14 14  2d
0811 : 14 14 42 41 53 49 43 20  b5
0819 : 43 4f 4e 44 45 4e 53 45  bf
0821 : 52 00 00 00 00 00 00 a2  b9
0829 : 56 a0 08 86 5f 84 60 a2  83
0831 : 13 a0 0f 86 5a 84 5b a2  a5
0839 : bd a0 c6 86 58 84 59 20  18
0841 : bf a3 a0 00 b9 13 0f 20  ab
0849 : d2 ff c8 c0 33 d0 f5 20  37
0851 : 44 a6 6c 02 a0 a9 0b 8d  e2
0859 : 20 d0 8d 21 d0 a2 ff a0  cc
0861 : c4 20 99 c0 20 d3 c4 a0  a9
0869 : 01 b1 2b d0 0a a2 6a a0  c8
0871 : c6 20 99 c0 6c 02 a0 a5  6a
0879 : 2d 8d 44 03 a5 2e 8d 45  6b
0881 : 03 a2 58 a0 c5 20 99 c0  45
0889 : a9 00 85 c6 20 e4 ff c9  29
0891 : 03 f0 3b c9 41 f0 04 c9  54
0899 : 42 d0 f1 8d 46 03 20 d2  14
08a1 : ff a2 8d a0 c6 20 99 c0  be
08a9 : a9 2a 8d 43 03 78 20 af  07
08b1 : c0 20 d3 c4 20 a8 c1 20  9d
08b9 : d3 c4 ad 46 03 c9 41 f0  88
08c1 : 0c 20 5b c3 20 d3 c4 20  21
08c9 : a8 c1 20 d3 c4 58 a2 a4  b7
08d1 : a0 c6 20 99 c0 ad 44 03  a0
08d9 : 38 e5 2d aa ad 45 03 e5  81
08e1 : 2e 20 cd bd a2 aa a0 c6  da
08e9 : 20 99 c0 6c 02 a0 86 7a  c8
08f1 : 84 7b a0 00 b1 7a f0 0b  24
08f9 : 20 d2 ff e6 7a d0 f5 e6  33
0901 : 7b d0 f1 60 a5 2b 85 7a  2b
0909 : a5 2c 85 7b 20 ef c4 a5  75
0911 : 7a 38 e9 01 85 41 a5 7b  32
0919 : e9 00 85 42 a0 04 20 71  3a
0921 : c4 a0 00 20 90 c1 b1 7a  0c
0929 : d0 03 4c 6f c1 c9 3a f0  b1
0931 : 46 d0 05 20 90 c1 b1 7a  f8
0939 : c9 00 d0 0e c8 c8 b1 7a  87
0941 : d0 01 60 88 20 71 c4 4c  f4
0949 : b7 c0 c9 3a d0 30 20 71  0c
0951 : c4 a5 7a 85 41 a5 7b 85  71
0959 : 42 a0 01 20 90 c1 b1 7a  03
0961 : f0 04 c9 3a d0 18 a5 41  f4
0969 : 85 7a a5 42 85 7b a0 01  96
0971 : 20 7c c4 a0 00 f0 bf c8  2d
0979 : 20 7c c4 4c cc c0 c9 22  d0
0981 : d0 1a c8 b1 7a f0 06 c9  a1
0989 : 22 d0 f7 f0 54 20 71 c4  c5
0991 : 20 a6 c4 a0 00 a9 22 91  42
0999 : 7a c8 d0 a0 c9 83 d0 20  fc
09a1 : c8 20 90 c1 b1 7a f0 94  b1
09a9 : c9 3a f0 a2 c9 2c f0 f0  c3
09b1 : c9 22 d0 09 c8 b1 7a f0  c6
09b9 : d4 c9 22 d0 f7 c8 d0 e4  e7
09c1 : c9 8f d0 1d a5 41 85 7a  99
09c9 : a5 42 85 7b a0 00 b1 7a  26
09d1 : d0 02 a0 04 c8 b1 7a d0  f1
09d9 : fb 20 7c c4 a0 00 4c e7  a7
09e1 : c0 c8 4c de c0 b1 7a c9  0b
09e9 : 20 d0 11 20 71 c4 a0 00  79
09f1 : a9 20 c8 d1 7a f0 fb 20  76
09f9 : 7c c4 a0 00 60 a9 00 8d  6e
0a01 : 40 03 a9 01 8d 3f 03 8d  47
0a09 : 41 03 20 8e a6 20 ef c4  5a
0a11 : a0 02 b1 7a d0 03 4c a2  0a
0a19 : c2 a5 7a 18 69 04 85 7a  11
0a21 : 90 02 e6 7b 20 73 00 a0  ba
0a29 : 00 b1 7a f0 e0 c9 22 d0  45
0a31 : 0e 20 73 00 c9 22 f0 ec  77
0a39 : c9 00 f0 d1 4c dc c1 c9  bf
0a41 : 89 f0 17 c9 8a f0 13 c9  51
0a49 : 8d f0 0f c9 a7 f0 0b c9  0d
0a51 : cb d0 d1 20 73 00 c9 a4  a4
0a59 : d0 cd 20 73 00 90 06 c9  b7
0a61 : 2c f0 f7 d0 c2 a5 7a a6  ae
0a69 : 7b 85 31 86 32 20 79 00  ce
0a71 : 20 6b a9 20 be c2 a0 01  3c
0a79 : b1 2f f0 ab c8 b1 2f 38  bb
0a81 : e5 14 c8 b1 2f e5 15 b0  b0
0a89 : 06 20 d3 c2 4c 21 c2 a5  11
0a91 : 7a e5 31 8d 3c 03 ad 3d  09
0a99 : 03 ae 3e 03 85 63 86 62  36
0aa1 : a2 90 38 20 49 bc 20 dd  54
0aa9 : bd a0 ff c8 b9 01 01 d0  18
0ab1 : fa cc 3c 03 f0 1f 90 0d  e5
0ab9 : 98 ed 3c 03 8d 42 03 20  ef
0ac1 : ef c2 4c 80 c2 ad 3c 03  c5
0ac9 : 8c 3c 03 38 ed 3c 03 8d  23
0ad1 : 42 03 20 1e c3 a0 00 b9  15
0ad9 : 01 01 f0 06 91 31 c8 4c  b6
0ae1 : 82 c2 ac 42 03 f0 03 20  3c
0ae9 : 33 a5 20 79 00 c9 2c f0  07
0af1 : 03 4c d2 c1 4c 05 c2 20  3f
0af9 : be c2 a0 01 b1 2f f0 12  dd
0b01 : c8 ad 3d 03 91 2f c8 ad  61
0b09 : 3e 03 91 2f 20 d3 c2 4c  57
0b11 : a5 c2 60 ad 3f 03 ae 40  2c
0b19 : 03 8d 3d 03 8e 3e 03 a5  c5
0b21 : 2b a6 2c 85 2f 86 30 60  04
```

```
0b29 : a0 01 b1 2f aa 88 b1 2f  b0
0b31 : 85 2f 86 30 ad 3d 03 18  f6
0b39 : 6d 41 03 8d 3d 03 90 03  ee
0b41 : ee 3e 03 60 a5 2d a6 2e  d6
0b49 : 85 5a 86 5b 18 6d 42 03  04
0b51 : 85 58 85 2d 90 01 e8 86  cb
0b59 : 59 86 2e a5 7a a6 7b 85  0b
0b61 : 5f 86 60 20 bf a3 a5 7a  c4
0b69 : 18 6d 42 03 85 7a 90 02  9b
0b71 : e6 7b 60 a5 2d a6 2e 38  13
0b79 : ed 42 03 b0 01 ca 85 2d  35
0b81 : 86 2e 38 a5 7a a6 7b ed  88
0b89 : 42 03 b0 01 ca 85 7a 86  69
0b91 : 7b 85 22 86 23 c5 2d 90  5e
0b99 : 04 e4 2e b0 d5 ac 42 03  83
0ba1 : b1 22 a0 00 91 22 a4 22  8c
0ba9 : c8 d0 01 e8 98 4c 3c c3  9b
0bb1 : a5 2b 85 7a a5 2c 85 7b  65
0bb9 : a9 00 85 3e 20 ef c4 a0  61
0bc1 : 03 c8 b1 7a f0 0f c9 22  d7
0bc9 : d0 24 c8 b1 7a f0 06 c9  ee
0bd1 : 22 d0 f7 f0 ec a5 3e f0  4e
0bd9 : 08 a9 fa 91 7a a9 00 85  a7
0be1 : 3e c8 c8 b1 7a f0 63 88  b9
0be9 : 20 71 c4 4c 67 c3 c9 fa  2e
0bf1 : f0 e3 c9 89 f0 18 c9 8d  88
0bf9 : f0 14 c9 9b f0 10 c9 8a  a5
0c01 : f0 0c c9 a7 f0 08 c9 8b  ec
0c09 : d0 b7 85 3e f0 b3 c8 20  ee
0c11 : 71 c4 a0 00 20 79 00 c9  6e
0c19 : 3a b0 a7 c9 30 90 a3 20  25
0c21 : 79 00 20 6b a9 20 13 a6  45
0c29 : b0 10 a0 00 b1 7a c9 2c  78
0c31 : d0 90 20 73 00 a0 ff 4c  5d
0c39 : b9 c3 a5 5f d0 02 c6 60  22
0c41 : c6 5f a0 00 a9 fa 91 5f  56
0c49 : d0 e0 a5 2b 85 7a a5 2c  73
0c51 : 85 7b 20 ef c4 a0 03 c8  89
0c59 : b1 7a f0 04 c9 fa d0 f7  ac
0c61 : c8 98 a0 00 91 7a c8 91  d1
0c69 : 7a a8 20 71 c4 a0 01 b1  26
0c71 : 7a d0 df a5 2b 85 7a a5  14
0c79 : 2c 85 7b a0 04 b1 7a f0  f4
0c81 : 07 c9 fa f0 36 c8 d0 f5  22
0c89 : 20 ef c4 84 3e c8 c8 b1  13
0c91 : 7a d0 01 60 18 65 3e 90  87
0c99 : 05 a4 3e 4c 66 c4 a5 7b  24
0ca1 : 85 40 a5 7a 18 65 3e 85  b0
0ca9 : 3f 90 02 e6 40 a0 04 20  e7
0cb1 : 84 c4 a4 3e a9 3a 91 7a  30
0cb9 : 4c 28 c4 a9 00 91 7a c8  88
0cc1 : 20 71 c4 4c 26 c4 98 18  70
0cc9 : 65 7a 85 7a 90 02 e6 7b  c8
0cd1 : 60 a5 7a 85 3f a5 7b 85  6d
0cd9 : 40 84 3d b1 3f a0 00 91  fd
0ce1 : 3f a4 3d e6 3f d0 f4 e6  bb
0ce9 : 40 a5 2e c5 40 b0 ec a5  c9
0cf1 : 2d 38 e5 3d 85 2d b0 02  e4
0cf9 : c6 2e 60 a5 2d 85 3f a5  ea
0d01 : 2e 85 40 e6 2d d0 02 e6  0e
0d09 : 2e a0 00 b1 3f c8 91 3f  bd
0d11 : a5 3f c5 7a d0 06 a5 40  6b
0d19 : c5 7b f0 0b a5 3f d0 02  d5
0d21 : c6 40 c6 3f 4c b4 c4 60  df
0d29 : 20 8e a6 a9 00 a8 91 7a  f0
0d31 : 20 33 a5 a5 23 85 2e 98  51
0d39 : 38 65 22 85 2d 90 02 e6  8a
0d41 : 2e 4c 60 a6 a9 9d 20 d2  30
0d49 : ff ad 43 03 49 0a 8d 43  f2
0d51 : 03 4c d2 ff 93 0e 08 9e  36
0d59 : 11 11 1d c2 c1 d3 c9 c3  fc
0d61 : 20 c3 cf ce c4 c5 ce d3  8e
0d69 : c5 d2 2c 20 31 39 38 34  cd
0d71 : 20 42 59 20 c3 48 2e 20  84
0d79 : da 57 45 52 53 43 48 4b  a1
0d81 : 45 1d 60 60 60 60 60 60  c4
0d89 : 60 60 60 60 60 60 60 60  89
0d91 : 60 60 60 60 60 60 60 60  91
0d99 : 60 60 60 60 60 60 60 60  99
0da1 : 60 60 60 60 60 60 60 60  a1
0da9 : 60 60 0d 11 00 41 29 20  8e
0db1 : cc 4f 45 53 43 48 54 20  e9
0db9 : 55 4e 57 49 43 48 54 49  8f
0dc1 : 47 45 20 cc 45 45 52 5a  c9
0dc9 : 45 49 43 48 45 4e 20 55  7f
0dd1 : 4e 44 0d 1d 1d 1d da 45  d9
0dd9 : 49 4c 45 4e 20 28 41 55  56
0de1 : 43 48 20 41 4c 4c 45 20  f5
0de9 : 22 d2 c5 cd 22 2d da 45  21
0df1 : 49 4c 45 4e 29 2e 0d 1d  ee
0df9 : 1d 1d ce 45 55 4e 55 4d  b9
0e01 : 45 52 49 45 52 55 4e 47  02
0e09 : 20 41 4c 4c 45 52 20 da  84
0e11 : 45 49 4c 45 4e 2e 0d 11  63
0e19 : 42 29 20 d7 49 45 20 41  b5
0e21 : 29 2c 20 41 42 45 52 20  68
0e29 : 44 49 45 20 da 45 49 4c  fd
0e31 : 45 4e 20 57 45 52 44 45  13
0e39 : 4e 20 5a 55 52 0d 1d 1d  15
0e41 : 1d 4d 41 58 49 4d 41 4c  fd
0e49 : 45 4e 20 cc 41 45 4e 47  5d
0e51 : 45 20 5a 55 53 41 4d 4d  f7
0e59 : 45 4e 47 45 46 41 53 53  a2
0e61 : 54 2e 0d 1d 1d 1d d6 4f  68
0e69 : 52 53 49 43 48 54 3a 20  70
0e71 : c4 41 53 20 d0 52 4f 47  1a
0e79 : 52 41 4d 4d 20 4b 41 4e  67
0e81 : 4e 20 44 41 4e 4e 0d 1d  de
0e89 : 1d 1d 4e 49 43 48 54 20  fa
0e91 : 4d 45 48 52 20 45 44 49  ad
0e99 : 54 49 45 52 54 20 57 45  5c
0ea1 : 52 44 45 4e 2e 0d 11 c2  46
0ea9 : 49 54 54 45 20 45 49 4e  c8
0eb1 : 47 45 42 45 4e 20 28 41  dd
0eb9 : 2f 42 29 20 3f 1d 00 cb  cc
0ec1 : 45 49 4e 20 c2 c1 d3 c9  60
0ec9 : c3 2d d0 52 4f 47 52 41  9c
0ed1 : 4d 4d 20 49 4d 20 d3 50  bc
0ed9 : 45 49 43 48 45 52 20 21  46
0ee1 : 0d 00 0d 11 c2 45 49 20  10
0ee9 : 44 45 52 20 c1 52 42 45  ab
0ef1 : 49 54 20 2e 2e 2e 20 2a  5b
0ef9 : 00 0d 11 d5 4d 20 00 20  95
0f01 : c2 59 54 45 53 20 47 45  0b
0f09 : 4b 55 45 52 5a 54 2e 0d  b6
0f11 : 09 00 0d 2a 2a 2a 20 42  9c
0f19 : 41 53 49 43 20 43 4f 4e  b5
0f21 : 44 45 4e 53 45 52 20 2a  c2
0f29 : 2a 2a 0d 0d 53 54 41 52  cf
0f31 : 54 45 4e 20 4d 49 54 20  70
0f39 : 22 53 59 53 20 34 39 31  b1
0f41 : 35 32 22 2e 0d 00 00 00  af
```

Listing 1. »Basic-Packer« – ein Programm, um Basic-Listings kürzer und schneller zu machen

HiRes-Dia-Show

Machen Sie aus Ihren HiRes-Grafiken eine professionelle Dia-Show! Das Besondere an diesem Programm ist das »weiche« Ein- und Ausblenden der einzelnen Bilder. Ihre Grafiken (zum Beispiel aus »Apfelmännchen«) sind somit »vorführungsreif«.

Dieses Programm (Listing 1) zeigt anhand einer Dia-Show mit den Grafiken von Apfelmännchen oder anderen HiRes-Grafiken die Möglichkeiten der Programme »HIRES-EFFEKT 1« und »HIRES-EFFEKT 2« (Listings 2 und 3). Das erste Programm wird dazu verwendet, um die hochauflösende Grafik auszublenden und damit zu löschen. Das zweite Programm »HIRES-EFFEKT 2« wird dazu verwendet, die Grafiken, die ständig von der Diskette nachgeladen werden, ineinander überzublenden, um einen fließenden Übergang zwischen den einzelnen Grafiken zu erhalten. Die auf dem Bildschirm sichtbare Grafik steht ab $2000, die Grafiken, die nachgeladen werden, ab $A000. Die Art, wie der Übergang vonstatten geht, wird von der RND(0)-Funktion gesteuert.

Im Sonderheft 2/86 wurde auf Seite 153/154 eine Idee vorgestellt, um die hochauflösende Grafik mit Hilfe des DIM-Befehls von Basic aus zu löschen. Darauf basierend kam mir der Einfall, diesen Einzeiler dahingehend zu erweitern, daß dieser nun auch dazu fähig ist, beliebige Bereiche des Speichers (in 255-Byte-Schritten) mit einem beliebigen Byte zu füllen. Dieser neue Einzeiler hat zwar äußerlich mit dem anderen nichts gemeinsam, benutzt aber die gleiche ROM-Routine zum Auffüllen des mit DIM dimensionierten Feldes mit dem Byte 0. Wenn man nun an geeigneter Stelle in das ROM springt, kann man sogar selbst bestimmen, mit welchem Byte der entsprechende Speicherbereich aufgefüllt werden soll. Der Einzeiler lautet folgendermaßen:

```
1 POKE88,0:POKE89,EndeHigh:POKE 113,0:POKE 114,
LängeHigh+1:POKE 780,Byte:POKE 12,1:SYS 45764
```

In den beiden Speicherzellen 88/89 wird im Low/High-Byte-Format die Endadresse des Bereiches angegeben, bis zu der man einen Speicherbereich mit einem Byte füllen möchte. In den beiden Speicherstellen 113/114 wird, ebenfalls im Low/High-Byte-Format, die Länge+1 des zu füllenden Speichers angegeben. In die Speicherstelle 780 schreibt man nun nur noch das Byte, mit dem der Speicherbereich ausgefüllt werden soll. (Die Speicherstelle 780 ist das A-Register des Prozessors, was dem aktuellen Stand des Akkumulators entspricht. Von Basic aus kann man diese Speicherstelle sowie die folgenden (genaue Belegung siehe Handbuch Seite 165) als LDA-Befehl beziehungsweise LDX, LDY und so weiter benutzen. Die Register werden nach dem SYS-Befehl automatisch belegt.

Als letzter POKE in dieser Zeile muß der POKE 12,1 stehen, der dieser Routine mitteilt, daß sie vom DIM-Befehl angesprungen wurde und auch kein Fehler aufgetreten ist. Der SYS-Befehl ist der Einsprungspunkt der Fill-Routine.

An ROM-Routinen wurde im Programm DIA-SHOW noch eine Routine zum Löschen einzelner Bildschirmzeilen verwendet. Um sie zu verwenden, wird vorher ins X-Register des Prozessors die Zeilennummer hinterlegt, die gelöscht werden soll (POKE 781, Zeilennummer; Zeilennummer darf zwischen 0 und 24 liegen) und daraufhin die Routine aufgerufen, die mit SYS 59903 zu starten ist. Studiert man das ROM-Listing dieser Routine, kann man damit sogar noch mehr machen:

Hinterlegt man im Y-Register (Speicherstelle 782) eine Zahl zwischen 0 und 39 und springt man die Routine mit SYS 59905 an, wird die im X-Register hinterlegte Zeile nur bis zu der dem Y-Register entsprechenden Spalte gelöscht!

Variablenbelegung des Programms

Die Variable A wird im Programm als Statusvariable für die Ladevorgänge gebraucht. Die Variablen E1, E2, L1, L2 und Byte sind mit der Beschreibung des Einzeilers hinreichend erklärt worden. Die Variablen I,Z,T sind reine Schleifenvariablen. Die Variable A$ dient der Speicherung des aktuellen Filenamens für den Ladevorgang der einzelnen Bilder.

Das Programm wird geladen und mit »RUN« gestartet.

Zunächst werden die notwendigen Routinen und die einzelnen Bilder nachgeladen und in den sichtbaren Bildschirm mit Hilfe der Routine »HIRES-EFFEKT 2« eingeblendet. Dieser Vorgang wird solange wiederholt, bis sämtliche auf der Diskette vorhandenen Bilder nachgeladen worden sind. Zum Schluß wird das letzte Bild mit Hilfe der Routine »HIRES-EFFEKT 1 ausgeblendet. Möchte man die DIA-SHOW für eigene Bilder umschreiben, muß in Zeile 7300 die Nummer des letzten zu ladenden Bildes eingetragen werden (IF I <Nummer THEN...). Der Name, den alle Bilder gemeinsam haben, steht in Zeile 7100 und kann ebenfalls nach Belieben geändert werden. (Für das erste Bild muß er auch in der Zeile 2000 geändert werden.) Die Bilder, die man nachlädt, sollten ab der Startadresse $A000 anfangen, dann braucht im Programm nichts gändert zu werden. Man kann die Grafiken aber auch in den $E000-Bereich verlegen. Ebenso ist es möglich, die Grafiken in allen 8K-Bereichen zu legen, die größer als $4000 sind. Die Startadresse dieser Grafik wird im ersten Parameter der Routine »HIRES-EFFEKT 2« dementsprechend geändert. Der SYS-Befehl zum Aufruf dieser Routine steht in der Zeile 7000. Auf jeden Fall müssen alle nachzuladenden Bilder die gleiche Startadresse haben! (Man kann sie bei Bedarf mit einem Diskmonitor ändern.)

(Christian Coppes/tr)

```
1800 IF A<3 THEN PRINT"{CLR,12DOWN,WHITE,7
     SPACE}BITTE EINEN MOMENT WARTEN !"CHR
     $(8)                                  <229>
1900 IF A=0 THEN A=1:LOAD"HIRES-EFFEKT 1",
     8,1                                   <136>
2000 IF A=1 THEN A=2:LOAD"BILD 1 .PIC",8,1 <058>
2100 IF A=3 THEN A=4:GOTO 6900             <146>
2200 IF A=4 THEN 7000                      <101>
2300 IF A=5 THEN A=6:GOTO 7500             <045>
2400 POKE 52,32:POKE 56,32                 <168>
2500 E1=0:E2=64:L1=0:L2=33:BYTE=255        <093>
2600 GOSUB 11600                           <248>
5300 POKE 781,12:SYS 59903                 <103>
5400 FOR I=0 TO 12                         <182>
5500 POKE 781,12+I:SYS 59903               <149>
5600 POKE 781,12-I:SYS 59903               <251>
5700 FOR T=1 TO 60:NEXT                    <091>
5800 NEXT                                  <222>
5900 POKE 53281,4                          <244>
6000 POKE 53265,59                         <191>
6100 POKE 53272,27                         <212>
6200 POKE 53270,216                        <210>
6300 E1=0:E2=220:L1=0:L2=5:BYTE=7          <111>
6400 GOSUB 11600                           <238>
6500 E1=0:E2=8:L1=0:L2=5:BYTE=2            <201>
6600 GOSUB 11600                           <182>
6700 SYS 828,8192,17,1                     <084>
6800 A=3:LOAD"HIRES-EFFEKT 2",8,1          <035>
6900 I=16                                  <187>
7000 SYS 828,40960,8192,I*INT(6*RND(0))+1  <098>
7100 A$="BILD"                             <023>
7200 I=I+1                                 <146>
7300 IF I<19 THEN A$=A$+STR$(I)+"*":LOAD A
     $,8,1                                 <100>
7400 A=5:LOAD"HIRES-EFFEKT 1",8,1          <143>
7500 SYS 828,8192,9,0                      <002>
7600 POKE 53281,0:POKE 53265,11:POKE 53270
     ,200:POKE 53272,21                    <116>
7650 POKE 198,0:WAIT 198,1:POKE 198,0      <131>
7700 POKE 2023,230:POKE 56295,5:POKE 53265
     ,27                                   <171>
11000 RESTORE:FOR I=1 TO 5:READ A          <026>
11100 E1=0:E2=220:L1=0:L2=5:BYTE=A         <151>
11200 GOSUB 11600                          <210>
11300 FOR T=1 TO 160:NEXT:NEXT             <187>
11400 DATA 1,15,12,11,0                    <052>
11500 PRINT"{CLR}":END                     <006>
11600 POKE 88,E1:POKE 89,E2:POKE 113,L1:PO
      KE 114,L2:POKE 780,BYTE:POKE 12,1:SY
      S 45764:RETURN                       <045>
```

Listing 1. Das Steuerprogramm »DIA-SHOW«

```
NAME : HIRES-EFFEKT 1  033C 03FC
----------------------------------
033C : 20 FD AE 20 EB B7 A5 15   48
0344 : 18 69 20 85 42 A5 14 8D   86
034C : AB 03 A5 15 8D AD 03 86   E4
0354 : 41 20 FD AE 20 9E B7 E0   92
035C : 00 F0 22 A9 25 8D B6 03   32
0364 : A9 77 85 02 20 A3 03 A9   E9
036C : 55 85 02 20 A3 03 A9 11   24
0374 : 85 02 20 A3 03 A9 00 85   FF
037C : 02 20 A3 03 60 A9 05 8D   5A
0384 : B6 03 A9 11 85 02 20 A3   79
038C : 03 A9 55 85 02 20 A3 03   20
0394 : A9 77 85 02 20 A3 03 A9   19
039C : FF 85 02 20 A3 03 60 A9   0A
03A4 : 00 8D AB 03 85 B4 A2 00   3F
03AC : A9 20 86 3F 85 40 A0 00   CC
03B4 : B1 3F 05 02 91 3F A5 3F   AF
03BC : 18 65 41 85 3F A5 40 69   7D
03C4 : 00 85 40 A5 3F C5 14 30   1E
03CC : 06 A5 40 C5 42 10 03 4C   B7
03D4 : B2 03 EE AB 03 AD AB 03   BB
03DC : C5 41 B0 11 46 B4 B0 0E   79
03E4 : B0 0C A5 02 0A 26 02 38   8E
03EC : 26 B4 4C AA 03 60 A5 02   A3
03F4 : 4A 66 02 06 B4 4C AA 03   11
```

Listing 2. »HIRES-EFFEKT 1«

```
NAME : HIRES-EFFEKT 2  033C 03F9
----------------------------------
033C : 20 FD AE 20 8A AD 20 F7   91
0344 : B7 A5 15 8D 95 03 20 FD   B3
034C : AE 20 EB B7 A5 15 18 69   32
0354 : 20 85 A5 A5 15 8D 8D 03   4F
035C : 86 A4 78 A9 34 85 01 A9   4E
0364 : 11 85 02 20 85 03 A9 55   7E
036C : 85 02 20 85 03 A9 77 85   11
0374 : 02 20 85 03 A9 FF 85 02   FD
037C : 20 85 03 A9 37 85 01 58   A9
0384 : 60 A9 00 8D 8B 03 A2 00   C6
038C : A9 20 86 AE 85 AF 86 AC   06
0394 : A9 A0 85 AD A0 00 A5 02   49
039C : 49 FF 31 AE 91 AE A5 02   30
03A4 : 31 AC 11 AE 91 AE A5 AE   C8
03AC : 18 65 A4 85 AE A5 AF 69   FA
03B4 : 00 85 AF A5 AC 18 65 A4   82
03BC : 85 AC A5 AD 69 00 85 AD   BE
03C4 : A5 AE C5 14 30 06 A5 AF   DD
03CC : C5 A5 10 03 4C 98 03 EE   3C
03D4 : 8B 03 AD 8B 03 C5 A4 B0   10
03DC : 11 46 B4 B0 0E B0 0C A5   35
03E4 : 02 0A 26 02 38 26 B4 4C   D5
03EC : 8A 03 60 A5 02 4A 66 02   D5
03F4 : 06 B4 4C 8A 03 4C AA 03   FC
```

Listing 3. »HIRES-EFFEKT 2«

Ein Hauch von Multitasking

Ein kleines Programm im Kassettenpuffer ermöglicht es, zwei Basic-Programme quasi gleichzeitig ablaufen zu lassen. Von jedem der zwei Programme wird abwechselnd je ein Befehl ausgeführt.

Tippen Sie das Programm »Fenchel MD« (Listing 1) mit dem MSE ein. Geladen wird es später mit »,8,1«. Der Aufruf erfolgt mit

```
SYS 828,Zeilennummer Programm 1, Zeilennummer Programm 2
```

Daraufhin wird von jedem Programm abwechselnd je ein Basic-Befehl ausgeführt. Das laufende Programm wird mit <←> gestoppt. Es geht zwar auch mit <RUN/STOP>, jedoch wird dann der geänderte Vektor nicht zurückgestellt. Hier noch ein Beispielprogramm:

```
10 sys828,100,200
100 print "programm1 ";
110 goto 100
200 print "programm2 "
210 goto 200
```

Beim Start mit RUN werden die Zeilen (100, 110) sowie (200, 210) als zwei eigenständige Programme betrachtet. Schritt für Schritt führt »Fenchel MD« nun jeweils abwechselnd einen Befehl von beiden Programmen aus. Die Reihenfolge ist also: 100, 200, 110, 210, 100, 200 etc.

Eine END-Anweisung in einem Programm führt zum Anhalten des gesamten Programmlaufes. Gibt man dann ein »CONT« ein, so wird das andere Programm bis zu seinem Ende abgearbeitet. Weiterhin dürfen sich die zwei Programme auch »überkreuzen«, wie es bei der gemeinsamen Benutzung von Unterprogrammen vorkommen kann. Doch hier ist Vorsicht geboten, wenn die Programme sich »überholen« (zum Beispiel durch IF). Denn dann stimmen die auf dem Stack abgelegten Rücksprungadressen der RETURNS nicht mehr. Meistens erscheint dann eine Fehlermeldung. Für Interessierte haben wir den Source-Code in Listing 2 abgedruckt und im untenstehenden Kasten erklärt.

(M. Dietz/og)

Erklärung der benutzten Speicherstellen

a/b	Zwischenspeicher für Zeilennummer #1
c/d	Zwischenspeicher für Zeilennummer #2
e	Flag, welches Programm abgearbeitet wird
$AEFD	Prüft auf Komma
$AD8A	Zahl oder Term holen, auf numerisch prüfen und in FAC ablegen
$B7F7	FAC in 2-Byte-Zahl wandeln
$A613	Startadresse einer Programmzeile berechnen und in $5F/$60 ablegen
$7A/$7B	Programmzeiger
$308/$309	Vektor für Basic-Befehlsadresse holen
$73	CHRGET-Routine
$A7ED	Basic-Befehl ausführen
198	Anzahl der Zeichen im Tastaturpuffer
$A480	Direktmodus ohne READY

Tabelle: Die von »Fenchel MD« benutzten Speicherstellen

```
programm : fenchel md          033c 03e5
-----------------------------------------

033c : 20 fd ae 20 8a ad 20 f7   91
0344 : b7 20 13 a6 38 a5 5f e9   a7
034c : 01 8d e5 03 a5 60 e9 00   f3
0354 : 8d e6 03 20 fd ae 20 8a   04
035c : ad 20 f7 b7 20 13 a6 38   b4
0364 : a5 5f e9 01 8d e7 03 a5   c3
036c : 60 e9 00 8d e8 03 a9 84   c9
0374 : 8d 08 03 a9 03 8d 09 03   c2
037c : a9 00 8d e9 03 4c 8b 03   8d
0384 : ae e9 03 d0 2f f0 0d ad   0c
038c : e5 03 85 7a ad e6 03 85   cd
0394 : 7b 4c ae a7 20 73 00 20   b4
039c : ed a7 a5 7a 8d e5 03 a5   75
03a4 : 7b 8d e6 03 ee e9 03 ad   a5
03ac : e7 03 85 7a ad e8 03 85   ff
03b4 : 7b 4c ae a7 20 73 00 20   d4
03bc : ed a7 a5 7a 8d e7 03 a5   a5
03c4 : 7b 8d e8 03 ce e9 03 a5   34
03cc : cb c9 39 f0 03 4c 8b 03   af
03d4 : a9 e4 8d 08 03 a9 a7 8d   8b
03dc : 09 03 a9 00 85 c6 4c 80   92
03e4 : a4 00 00 00 00 00 00 00   89
```

Listing 1. »Fenchel MD« geben Sie bitte mit dem MSE ein

```
        .OPT P4
        *=   828
A       =    $3E5
B       =    $3E6
C       =    $3E7
D       =    $3E8
E       =    $3E9
        JSR  $AEFD
        JSR  $AD8A
        JSR  $B7F7
        JSR  $A613
        SEC
        LDA  $5F
        SBC  #01
        STA  A
        LDA  $60
        SBC  #00
        STA  B
        JSR  $AEFD
        JSR  $AD8A
        JSR  $B7F7
        JSR  $A613
        SEC
        LDA  $5F
        SBC  #01
        STA  C
        LDA  $60
        SBC  #00
        STA  D
        LDA  #$84
        STA  $308
        LDA  #$03
        STA  $309
        LDA  #00
        STA  E
        JMP  W1
        LDX  E
        BNE  Q1
        BEQ  Q2
        LDA  A
        STA  $7A
W1      LDA  B
        STA  $7B
        JMP  $A7AE
        JSR  $73
        JSR  $A7ED
Q2      LDA  $7A
        STA  A
        LDA  $7B
        STA  B
        INC  E
        LDA  C
        STA  $7A
        LDA  D
        STA  $7B
        JMP  $A7AE
        JSR  $73
        JSR  $A7ED
Q1      LDA  $7A
        STA  C
        LDA  $7B
        STA  D
        DEC  E
        LDA  203
        CMP  #57
        BEQ  Q3
        JMP  W1
Q3      LDA  #$E4
        STA  $308
        LDA  #$A7
        STA  $309
        LDA  #00
        STA  198
        JMP  $A480
```

Listing 2. Source-Code zu »Fenchel MD«



Basic-Maker: Ein Recompiler für die gängigsten Compiler

Von jetzt an können Sie getrost Ihre Basic-Programme nach dem Compilieren löschen, denn jetzt gibt es Basic-Maker. Basic-Maker generiert aus jedem Compilat, das mit den Austro-Compilern erzeugt wurde, ein lauffähiges und editierbares Basic-Programm.

Ein Recompiler ist ein Übersetzer, der es ermöglicht, ein compiliertes Programm wieder in das Ursprungs-Basic-Programm zurückzuverwandeln. Da jeder Basic-Compiler unterschiedlichen Code erzeugt, kann man mit einem bestimmten Recompiler immer nur Compilate dieses Compilertyps verarbeiten.

Basic-Maker ist in der Lage, Compilate der in Deutschland am meisten verbreiteten Compiler zu verarbeiten, des Austro-Comps und dessen Nachfolgern Austro-Speed sowie Austro-Blitz. Diese Compiler unterscheiden sich nur geringfügig im Aufbau und erzeugen somit sehr ähnliche Compilate, die von Basic-Maker bearbeitet werden.

Zunächst scheint es unsinnig zu sein, einen Recompiler zu schreiben, da man ja gerade durch das Compilieren ein schnelleres und meistens auch kürzeres Programm erhalten hat. Man nimmt jedoch auch einen gewaltigen Nachteil in Kauf: Das Programm ist für immer in der vorliegenden Form fixiert, es können also keinerlei Änderungen mehr vorgenommen werden. Will man sich Weiterentwicklungen offenhalten, muß man neben dem Compilat auch noch das Ursprungsprogramm speichern. Darauf verzichten jedoch die meisten Anwender, weil Diskettenspeicherplatz nicht gerade billig ist. Basic-Maker ermöglicht es durch die Übersetzung des Compilats in Basic-Quelltext, doch noch Änderungen am Programm vorzunehmen. Anschließend kann man das geänderte Programm wieder compilieren.

Besonders tragisch sind die Fälle, in denen im Compilat eines Basic-Programms Fehler auftreten. Diese Programme waren bisher als Totalverlust zu werten, durch den Einsatz von Basic-Maker kann man das Programm jedoch noch retten.

Die Bedienung von Basic-Maker

Durch Basic-Maker ist es ebenso möglich geworden, gekaufte Software, die mit den Austro-Compilern compiliert wurde, nach eigenen Wünschen abzuändern und damit sogenannte »Schwachstellen« des Programms auszubessern. Ich habe zum Beispiel den Pascal-Compiler »PASCAL 64« recompiliert und so abgeändert, daß das Programm jetzt nach erneuter Compilierung um fast 25 Prozent schneller arbeitet.

Der Start der Übersetzung

Wenn Sie das Programm (Listing) mit RUN gestartet haben, sehen Sie das umfangreiche Menü auf dem Bildschirm. Die Recompilierung wird gestartet, indem man in den Menüpunkten 1 bis 3 den Typ des Compilers festlegt, mit dem das nun zu bearbeitende Compilat erstellt wurde. Nach dieser Wahl erscheint die Frage nach dem Namen des Compilats auf Diskette. Wenn Sie diesen eingegeben haben, beginnt die Übersetzung.

Die Recompilierung: PASS 1

Bei Basic-Maker handelt es sich um einen 2-Pass-Recompiler, der das Compilat also in zwei Arbeitsgängen übersetzt. In PASS 1 werden dabei die Zeilennummern für das Basic-Programm festgelegt, die Variablen geholt sowie grobe Aufbaufehler gesucht, die eine korrekte Übersetzung verhindern würden. Im Gegensatz zu allen bekannten Compilern werden diese Informationen nicht in eine Hilfsdatei auf Diskette, sondern in ein spezielles Feld im Speicher geschrieben. Dadurch wird der Zugriff erheblich beschleunigt. Das Feld befindet sich im Adreßraum von $A000 bis $FFFF, was einer Kapazität von 24 KByte entspricht. Damit lassen sich Compilate von einer Länge bis zirka 115 Blöcke recompilieren (das längste, mir bekannte Compilat ist das Programm »UNI-TAB« mit einer Länge von 106 Blocks).

Die Recompilierung: PASS 2

Die eigentliche Übersetzung wird von Basic-Maker in PASS 2 vorgenommen. Dabei wird der Code in Basic-Befehle umgewandelt und in einzelne Programmzeilen geschrieben. Das erzeugte Basic-Programm wird parallel auf dem Bildschirm gelistet und in eine Diskettendatei geschrieben, so daß Sie es mit `LOAD"Name",8` laden können. Als Programmname wird der Name des Compilats mit dem Vorsatz »B—« verwendet. Es wird ein zum Ursprungsprogramm völlig ablaufkompatibles Programm erzeugt, das sich also im Ablauf gleich verhält. Dabei wird das Programm aus folgenden Gründen jedoch nie genau gleich zum Ursprungsprogramm sein: REM-Zeilen werden bei der Compilierung entfernt, da sie im Code sinnlos sind. Daher kann das Recompilat ebenso keine REM-Zeilen enthalten. Im Gegensatz zum Basic-Programm enthält das Compilat ebenso keine Zeilennummern, da der Code chronologisch vom Runtime-Modul des Compilers abgearbeitet wird. Es läßt sich nachträglich nicht mehr feststellen, welche Zeilennummern das Ursprungsprogramm enthielt und welche Befehle in welchen Zeilen standen. Basic-Maker beginnt daher normalerweise mit der Zeile 1 und erzeugt dann pro Programmzeile nur einen Befehl, damit das Basic-Programm leicht editierbar wird. Eventuell auftretende DATA-Codes werden vom Compiler an einer Stelle des Compilats »gesammelt«. Da die Position der DATA-Zeilen im Basic-Programm für die Reihenfolge der Bearbeitung unerheblich ist, bleibt dies ohne Folge für den Programmablauf. Basic-Maker kann natürlich nicht feststellen, woher die einzelnen DATAS kommen und hängt diese komplett an das erzeugte Basic-Programm an. Dadurch wird zwar das Aussehen, nicht aber die Funktion des Programms beeinflußt.

Nach Beendigung der Übersetzung werden die Anzahl der aufgetretenen Fehler und Erweiterungen sowie der Name

des erzeugten Basic-Programms ausgegeben. Anschließend können Sie durch Drücken der Taste <N> einen Reset auslösen oder durch eine beliebige andere Taste Basic-Maker erneut starten.

Die Befehle MID$, RUN, PRINT$ und die FOR-NEXT-Schleife

Im Gegensatz zu den anderen Basic-Befehlen werden die oben genannten bei der Übersetzung durch den Compiler aus verschiedenen Gründen verändert. Der MID$-Befehl tritt normalerweise nur in Verbindung mit drei Parametern auf, wie zum Beispiel MID$(A$,2,3). Er kann aber auch mit nur zwei Parametern verwendet werden, zum Beispiel MID$(A$,2). Die Compiler können durch den Aufbau des Runtime-Moduls jedoch nur Befehle mit drei Parametern verwenden. Deshalb wird bei der Compilierung eine 255 als dritter Parameter angehängt. Die Wirkung des Befehls wird nicht verändert, da 255 die maximal zulässige Stringlänge ist. Bei der Recompilierung erzeugt Basic-Maker daher in jedem Fall einen MID$-Befehl mit drei Parametern.

Der RUN-Befehl wird bei der Compilierung durch zwei andere Basic-Befehle ersetzt, so daß keine eigene Interpretationsroutine für ihn erforderlich ist. Diese Befehle sind CLR und GOTO Z (Z: Erste Zeile des Basic-Programms). Durch sie wird genau das erreicht, was auch der RUN-Befehl bewirkt hätte. Basic-Maker »weiß« bei der Übersetzung nicht, ob diese Befehlsfolge einen »RUN«-Befehl ersetzt hat oder in der Form im Ursprungsprogramm vorhanden war. Deshalb wird sie unverändert in das Recompilat übernommen.

Für den PRINT#-Befehl besitzen die Compiler ebenso keine Interpretationsroutine. Beim Antreffen dieses Befehls geschieht deshalb folgendes: Zunächst wird der PRINT#-Befehl durch die Befehle CMD und PRINT ersetzt, wodurch die Ausgabe auf das angesprochene Gerät umgelenkt wird. Anschließend muß die Ausgabe auf den Bildschirm zurückgesetzt werden. Dafür benutzen die Compiler jedoch keinen Basic-Befehl. Basic-Maker fügt daher beim Antreffen der Befehlsfolge CMD und PRINT von sich aus einen PRINT#-Befehl an. Dadurch wird die Ausgabe wieder auf den Bildschirm geleitet.

Eine Schleife wird normalerweise in die Form »FOR I = Anfangswert TO Endwert:...:Next I« benutzt. Die Compiler ändern diese jedoch in die Form »I = Anfangswert:FOR I = I TO Endwert:...:NEXT I« ab. Die Wirkung bleibt unverändert. Basic-Maker übernimmt die Schleifenform des Compilats in das erzeugte Basic-Programm, da auch hier nicht festgestellt werden kann, ob dies ein Erzeugnis des Compilers ist oder vom Ursprungsprogramm stammt.

Der erweiterte SYS-Befehl

Die Austro-Compiler zeichnen sich unter anderem dadurch aus, jede Form des SYS-Befehls verarbeiten zu können. Basic-Maker ist in der Lage, diese erweiterten Befehle zurückzuübersetzen. Durch den Aufbau des Programms werden jedoch meistens Fehlermeldungen beim Antreffen eines erweiterten SYS-Befehls ausgegeben. Das hat jedoch hier keine Bedeutung. Der Befehl wird in jedem Fall korrekt übersetzt, so daß das Basic-Programm einwandfrei arbeiten kann.

Die Fehlermeldungen von Basic-Maker

Basic-Maker gibt zwei Arten von Fehlermeldungen aus: Zum einen gibt es die »leichten« Fehler, die in der Form »? RECOMPILE ERROR ?« mit Fehleradressen ausgegeben werden. Diese Fehlermeldungen werden durch den Aufbau von Basic-Maker bedingt auch bei Erweiterungen oder erweiterten SYS-Befehlen ausgegeben, ohne daß tatsächlich ein Fehler vorliegt. Meistens bedeutet ein »leichter« Fehler, daß Basic-Maker einen unbekannten Code gefunden hat. Wenn eine nicht interpretierbare Programmzeile gefunden wird, so wird in dem recompilierten Basic-Programm eine REM-Zeile erzeugt. Sie erkennen daher beim Listen dieses Programms sofort die fehlerbehafteten Stellen, da ja sonst keine REM-Zeilen existieren. Die Übersetzung kann in fast allen Fällen nach einem »leichten« Fehler korrekt fortgesetzt werden.

Im Gegensatz dazu können auch schwerwiegende, sogenannte »fatale« Fehler auftreten. Diese Fehler treten meist in PASS 1 auf und deuten auf einen schwerwiegenden Aufbaufehler des Compilats hin. Meistens treten »fatale« Fehler nur dann auf, wenn das Compilat nicht korrekt erzeugt wurde, das heißt wenn während der Compilierung von den Austro-Compilern Fehlermeldungen ausgegeben wurden. Basic-Maker kann nach dem Auffinden eines »fatalen« Fehlers nicht mehr korrekt weiterarbeiten und bricht die Recompilierung ab. Dabei werden alle Diskettendateien korrekt geschlossen, so daß eventuell bis zu diesem Zeitpunkt korrekt übersetzte Teile des Compilats nicht verlorengehen. Tritt innerhalb des Basic-Maker-Recompilers ein Fehler auf (zum Beispiel wenn das Programm im Direktmodus gestartet wurde), wird die Meldung »SYSTEM ERROR« ausgegeben und die Übersetzung ebenso abgebrochen. Basic-Maker liest zur Sicherheit nach jedem Zugriff auf die Diskettenstation den Fehlerkanal aus. Wird dabei eine Fehlermeldung gefunden, wird die Recompilierung sofort abgebrochen, da auf keinen Fall korrekt weitergearbeitet werden kann. Auch hier werden vor dem Abbruch alle Dateien richtig geschlossen.

Basic-Maker für Compiler-Enthusiasten

Nachdem grundsätzlich geklärt ist, wie man mit Basic-Maker eine Recompilierung vornehmen kann, soll jetzt auf die Menüpunkte 4 bis 9 eingegangen werden.

Festlegen der Startzeile und Schrittweite

Bisher wurde immer davon ausgegangen, daß Basic-Maker das erzeugte Basic-Programm mit der Zeilennummer 1 beginnt und in Einer-Schritten fortsetzt. Im Menüpunkt 4 können Sie jedoch eine andere Startzeile/Schrittweite festlegen. Erlaubt sind Werte von 1 bis 10. Aus programmtechnischen Gründen muß die Schrittweite immer der Startzeile des Basic-Programms entsprechen. Bei jedem Druck der Taste <4> wird die Zahl um 1 erhöht, bis 10 erreicht ist. Daraufhin wird wieder mit 1 begonnen, und so weiter. Wenn Sie also vorhaben, viele Zeilen in das erzeugte Basic-Programm einzufügen, sollten Sie lieber eine etwas größere Schrittweite festlegen.

Wahl der Bildschirmausgabe

In Punkt 5 können Sie bestimmen, ob das recompilierte Programm in PASS 2 auf dem Bildschirm ausgegeben werden soll. Wenn Sie nichts anderes festlegen, wird das Listing immer auf dem Bildschirm ausgegeben. Bei äußerst langen Compilaten kann jedoch durch Abschalten der Bildschirmausgabe die Verarbeitungsgeschwindigkeit von Basic-Maker in PASS 2 um mehr als 35 Prozent erhöht werden. Wenn man daher nicht unbedingt an einer Ausgabe interessiert ist, kann man durch Drücken der Taste <5> die Bildschirmausgabe abschalten. Fehlermeldungen und Erweiterungen werden jedoch in jedem Fall weiterhin ausgegeben. Durch erneute Betätigung der Taste wird die Ausgabe immer wechselweise ein- und ausgeschaltet. Der jeweilige Zustand wird hinter der Menü-Zeile angezeigt. Bei kleinen Compilaten ist der Geschwindigkeitsgewinn nicht lohnenswert, so daß man die Ausgabe eingeschaltet lassen kann.

Bestimmung der Länge einer DATA-Zeile

Im Menüpunkt 6 können Sie die maximale Länge einer erzeugten DATA-Zeile festlegen. Die Voreinstellung beträgt fünf Zeichen, wobei alle Werte ohne Zeilennummer und Befehlswort »DATA« gerechnet werden, so daß wirklich nur die Länge der eigentlichen DATAs betrachtet wird. Durch Drücken der Taste <6> wird diese Einstellung jeweils um

fünf Zeichen erhöht. Das Maximum beträgt durch die maximale Zeilenlänge (80 Zeichen) 70 Zeichen. Besonders bei Compilaten mit vielen DATAs kann man durch lange DATA-Zeilen eine erhebliche Anzahl von Programmzeilen einsparen. Wird ein DATA-Statement gefunden, dessen Länge die voreingestellte, maximale übertrifft, wie zum Beispiel eine Stringkette, so wird diese Voreinstellung natürlich ignoriert, das heißt das DATA-Element wird korrekt erzeugt.

Die Recompilierung von Basic-Erweiterungen

Die Austro-Compiler sind vor allen Dingen deshalb so beliebt, weil sie beliebige Basic-Befehlserweiterungen verarbeiten können. Basic-Maker ist in der Lage, verwendete Erweiterungsbefehle korrekt zu recompilieren. Es ist jedoch unbedingt erforderlich, den verwendeten Erweiterungstyp in den Menüpunkten 7 bis 9 festzulegen.

Voreingestellt ist der Punkt 7, das heißt keine Erweiterung. Durch Punkt 8 wird eine Erweiterung festgelegt, deren Befehle ausschließlich 1-Byte-Token sind. Der Commodore 64 verwendet bekanntlich die Token 128 bis 203 und das Token 255, so daß für eine Befehlserweiterung noch die Token 204 bis 254 zur Verfügung stehen. Haben Sie den Punkt 8 angewählt, so müssen Sie Basic-Maker das Start-Token (204 voreingestellt) und End-Token (254 voreingestellt) der Erweiterung angeben. Es ist ganz wichtig, daß Sie diese Frage korrekt beantworten, da Basic-Maker sonst nicht immer ein korrekt arbeitendes Basic-Programm erzeugen kann. Nach der Eingabe dieser Token wird hinter der Menü-Zeile angezeigt, daß Sie eine 1-Byte-Token-Erweiterung verwendet haben. Ein Beispiel für eine Basic-Erweiterung, die die Token 204 bis 254 benutzt, ist »GRAFIK-BASIC« aus der Zeitschrift »RUN Spezial Nr. 1«. Neben dieser Erweiterungsform gibt es jedoch auch Programme, die 2 Byte als Token verwenden. Dies ist immer dann erforderlich, wenn mehr als 51 neue Befehle vorhanden sind, da durch 1-Byte-Token ja nur maximal 51 solcher codiert werden können. Deshalb verwenden größere Erweiterungen immer das 2-Byte-Token-Prinzip: Dabei wird jedem neuen Befehl ein sogenanntes Erkennungs-Token vorangestellt, das nur die Funktion hat, anzuzeigen, daß ein Erweiterungsbefehl folgen wird. Dieses Erkennungs-Token ist bei allen Erweiterungsbefehlen identisch. Erst dann folgt das eigentliche Befehls-Token. Durch dieses Prinzip, was gegenüber dem 1-Byte-Token natürlich umständlicher ist, kann man auch die schon vom Commodore 64 verwendeten Token belegen und damit natürlich weit mehr Befehle codieren. Bei Punkt 9 des Menüs wird daher nur nach dem Erkennungs-Token der Erweiterung gefragt, die das Compilat verwendet. Auch hier gilt, daß bei falscher Eingabe eine korrekte Übersetzung unmöglich ist. Als Beispiel möchte ich »Simons Basic« mit dem Erkennungs-Token 100 anführen. Zu der Gruppe der 2-Byte-Token-Erweiterungen gehören übrigens auch diejenigen Erweiterungen, die vor jedem Befehl ein Ausrufezeichen oder ähnliches stehen haben und das Befehlswort uncodiert benutzen. Im Fall des Ausrufezeichens wäre das Erweiterungs-Token die 33 (ASCII-Code von »!«). Durch den Aufbau des Basic-Maker-Recompilers werden bei der Übersetzung von Erweiterungsbefehlen immer Fehlermeldungen erzeugt. Wie beim erweiterten SYS-Befehl bleiben diese jedoch ohne Auswirkung auf das recompilierte Basic-Programm.

Wie man Erweiterungen erkennt

Zunächst scheint es schwierig zu sein, den Typ einer Erweiterung festzustellen. Eindeutig ist der Fall bei Erweiterungen, die vor jedem Befehl ein bestimmtes Zeichen haben, zum Beispiel ein Ausrufezeichen. Hierbei handelt es sich immer um den 2-Byte-Token-Typ. Ist den Befehlen jedoch kein Zeichen vorangestellt, müssen Sie experimentieren. Dafür schreiben Sie am besten ein kleines Basic-Programm, was zirka fünf Erweiterungsbefehle benutzt. Jeder Befehl muß in einer neuen Programmzeile stehen. Dann sehen Sie sich mit einem Monitor den Basic-Speicher ab $0801 an. Wenn jede Programmzeile mit demselben Code beginnt, obwohl verschiedene Befehle verwendet wurden, handelt es sich in jedem Fall um eine 2-Byte-Token-Erweiterung. In einem anderen Fall kann es sich nur um eine 1-Byte-Token-Erweiterung handeln. Nun müssen Sie nur noch die verwendeten Befehls-Token feststellen: Geben Sie dazu im Direktmodus die folgenden Befehle ein:

```
NEW (Return)
POKE 2049,6:POKE 2050,8 (Return)
POKE 2051,10:POKE 2052,0 (Return)
POKE 2054,0:POKE 2055,0:POKE 2056,0 (Return)
POKE 45,10:POKE 46,8:CLR: I=203 (Return)
```

Nun müssen Sie immer folgende Zeile eingeben, bis I den Wert 254 erreicht hat:

```
I=I+1:PRINT I:PRINT"(Down,10Space,2Up)":POKE 2053,
I:LIST (Return)
```

Wenn nach dem List-Befehl ein Befehlswort der Erweiterung gelistet wird, ist der Wert I ein 1-Byte-Token. Ist dies nicht der Fall, so wird nur das ASCII-Zeichen von I ausgegeben, das heißt I ist kein verwendetes Token. Auf diese Weise können Sie genau die verwendeten Token der Erweiterung bestimmen. Sie müssen die obige Zeile dabei immer mit dem Cursor überfahren und »RETURN« drücken. Das Start-Token stellt dabei den ersten ausgegebenen Wert von I dar, der als Befehlswort gelistet wird. Das End-Token der Erweiterung ist dann erreicht, wenn kein Befehl, sondern nur noch ein ASCII-Zeichen gelistet wird, also der ausgegebene Wert von I minus 1.

(Frank Riemenschneider/ah)

```
name : basic-maker      0801 3cc8
-------------------------------------
0801 : 0e 08 ca a8 9e 32 30 36  84
0809 : 35 20 46 43 43 00 00 00  7d
0811 : a0 00 b9 69 07 99 00 cd  26
0819 : b9 69 08 99 00 ce b9 69  ec
0821 : 09 99 00 cf c8 d0 eb 4c  4c
0829 : c2 cd 78 a0 ff 84 fb a9  6b
0831 : c6 85 fc a9 36 85 01 8d  dd
0839 : 20 d0 c8 a5 2d d0 02 c6  97
0841 : 2e c6 2d a6 2e e0 0a d0  a6
0849 : 04 c9 67 f0 0f b1 2d 91  80
0851 : fb a5 fb d0 02 c6 fc c6  10
0859 : fb 4c d3 cd a2 08 a9 01  3c
0861 : 86 2e 85 2d 84 ff 20 50  6f
0869 : ce c9 f3 d0 27 20 50 ce  85
0871 : aa 86 fa c9 04 b0 04 a9  7f
0879 : f3 d0 03 20 50 ce a0 00  97
0881 : 91 2d c8 c6 fa d0 f9 98  03
0889 : 18 65 2d 85 2d 90 02 e6  7d
0891 : 2e 4c 34 ce a0 00 91 2d  77
0899 : e6 2d f0 f3 a9 45 a2 41  a2

08a1 : e4 2e d0 c2 c5 2d d0 be  af
08a9 : a9 37 85 01 a9 fe 8d 20  78
08b1 : d0 58 20 59 a6 4c ae a7  b7
08b9 : a2 ff 86 f7 86 f8 e8 a9  22
08c1 : 01 85 fe a9 7f 85 fd c6  23
08c9 : ff 10 10 e6 fb d0 02 e6  cd
08d1 : fc a9 07 85 ff a0 00 b1  7d
08d9 : fb 85 f9 06 f9 b0 0a a4  6d
08e1 : fe a5 fd 39 f7 00 99 f7  2e
08e9 : 00 8a 0a a8 a5 f7 38 f9  b5
08f1 : e2 ce a5 f8 f9 e3 ce 90  de
08f9 : 0e e0 0d f0 0a e8 38 66  6e
0901 : fd b0 c4 c6 fe f0 bc 8a  e0
0909 : f0 0f a5 f7 38 f9 e0 ce  5e
0911 : 85 f7 a5 f8 f9 e1 ce 85  0f
0919 : f8 a4 fe f0 07 a5 f8 85  ce
0921 : f7 88 84 f8 a5 fd 4a 90  31
0929 : 07 46 f8 66 f7 4c be ce  d9
0931 : bd d2 ce 65 f7 a8 b9 00  63
0939 : cf 60 00 00 00 00 01 04  45
0941 : 14 34 65 91 bf de f4 fa  b8
0949 : fe fe 00 00 00 00 00 00  c7

0951 : 00 10 00 28 00 68 00 a8  f3
0959 : 00 d9 00 ef 80 fa 60 fe  a3
0961 : c0 ff f0 ff 00 00 00 00  5d
0969 : 9f 01 22 00 85 20 27 a9  5b
0971 : a5 51 37 54 57 28 1a 88  47
0979 : 3a b0 2e 4c 1e 45 47 60  0a
0981 : ab 6d 48 49 2c 02 8b 3d  9a
0989 : 8a 3c 84 0d 39 53 89 29  9b
0991 : 26 24 03 a6 2f 18 3b 4e  9c
0999 : 58 52 3f 0c 16 1c d0 67  c0
09a1 : 0f 68 ff 10 38 08 2d 40  df
09a9 : 4f 61 f0 8d 0a 43 4b a0  c0
09b1 : 0b 41 09 90 69 b1 34 86  b3
09b9 : 0e 46 3e 44 1f 83 17 04  75
09c1 : 33 30 32 e5 12 91 c8 e8  f8
09c9 : 50 ca 4d d8 fb a4 b5 aa  fe
09d1 : 05 13 6c a7 bd 4a a8 9e  7e
09d9 : 15 2b 42 63 92 f3 95 e9  74
09e1 : 56 65 23 9c 9d 19 e2 5a  29
09e9 : 80 06 1b a2 14 59 11 c9  6b
09f1 : ba 8e dd 62 6e 25 77 7f  a3
09f9 : 1d a1 55 66 79 98 07 36  ee
```

```
0a01 : 74 7e d3 e6 ad b2 ec 35  14
0a09 : 5e 7b 97 bc 31 6a 6f ce  64
0a11 : 73 7a 99 c6 ed 21 b9 bb  47
0a19 : f6 2a 71 78 f9 5b 5c 64  44
0a21 : be c5 f1 fe 5d 5f 75 bf  44
0a29 : c0 76 82 e0 7c 8c b8 d9  a4
0a31 : 70 c2 da e1 ef 81 8f 94  67
0a39 : 9a af b4 b6 e7 eb f7 f8  5e
0a41 : b7 d7 7d 87 a3 b3 c4 c7  af
0a49 : cc d1 d5 df fa 96 c3 cd  7e
0a51 : d4 db dc f2 fc fd 6b 72  fa
0a59 : 93 ae c1 d6 f5 9b ac cf  1d
0a61 : d2 e3 e4 de ea f4 a9 b1  9a
0a69 : fb de 0e f3 3b 35 0b 02  63
0a71 : f7 f7 b1 5d dc d9 6b 5e  83
0a79 : dd 35 a3 c2 da 96 42 13  c3
0a81 : 6f 93 dd 66 32 94 fc ef  9a
0a89 : cf f5 28 79 d8 9f fd f8  00
0a91 : b9 7e bf d4 ff fb d9 f3  43
0a99 : 7e c4 4f 3b fd b9 7e f5  48
0aa1 : 4f e7 e5 d7 ca 33 bb fd  89
0aa9 : 1e 76 cd c5 5f 77 07 9f  3b
0ab1 : f6 6f bf 1d f7 42 fb 67  43
0ab9 : 7d ce a9 f8 ef b5 6f bc  0b
0ac1 : 5b ef f2 be f5 ef be 6b  59
0ac9 : ea cb 8d ed c5 4d c4 9b  cb
0ad1 : 8d f5 c6 65 c2 d7 18 f7  f2
0ad9 : 17 57 08 dc 73 2e 1b dc  08
0ae1 : 55 dc 7a 77 1d cb 85 ee  56
0ae9 : 3f c6 e1 ed e5 e5 e7 46  7b
0af1 : f3 e3 e7 a9 79 ba bc fa  5b
0af9 : d7 9c 3b cd 95 e6 5d 3b  23
0b01 : 5a 76 f4 fe 3a 7f 82 9f  9c
0b09 : 2a 1a 10 f7 f0 fe b4 3f  9c
0b11 : 36 1f 4f 13 23 13 de c4  dd
0b19 : 2b 13 83 88 18 9e 3e 27  7d
0b21 : 57 13 f9 f1 3f c3 13 f0  ff
0b29 : 62 7e 1c 4a ca ca 8a c9  db
0b31 : 95 92 2b 32 2b 3c 2a c6  eb
0b39 : 6b 12 ac ec 56 72 eb 3f  9d
0b41 : f1 59 a9 5f f4 57 fa 95  56
0b49 : e1 5f fa ba 7f e7 d3 f7  66
0b51 : fa 6f 74 2f 34 34 34 0c  d4
0b59 : d0 9d a1 f3 e8 76 74 3e  6f
0b61 : df 3f c4 d0 fe 1a 1c 1d  96
0b69 : 00 d0 fe ad 0f 5f 43 93  67
0b71 : cf 16 d8 cd b6 ff 6d c9  f0
0b79 : db 6c b6 df bb 6d 4b a2  cd
0b81 : cf 45 bf 47 c3 e8 ab d1  a1
0b89 : f0 f7 be 66 f7 c0 de 9d  2d
0b91 : bd e2 6f 76 b5 1f c2 a3  11
0b99 : bc a8 f9 aa 05 95 71 95  6b
0ba1 : fb b6 9c fc ae ee 52 d9  1d
0ba9 : 5b ac ae 6e 56 cb 2b ff  44
0bb1 : 59 5f 6f 2b 84 2c a9 b3  b3
0bb9 : 71 e6 f6 26 ff 29 bf 7a  5d
0bc1 : 6f a5 37 95 ca c6 e5 5e  bb
0bc9 : f2 99 e5 10 9d 9e 57 fd  2b
0bd1 : 72 bf cb 95 f8 d4 fd 8a  0c
0bd9 : 7f aa 8b 73 e5 f3 f4 b9  44
0be1 : 5c 68 6c d4 c5 a9 f2 2a  f1
0be9 : 7f 9a a7 fe ea 7f 55 4f  1e
0bf1 : eb a9 d0 ee e4 77 4b c5  85
0bf9 : a6 54 38 58 97 64 c1 86  93
0c01 : 53 6d 2c 18 65 3a 6e 5e  b8
0c09 : 2d 32 a1 c2 c4 bb 26 0c  eb
0c11 : 32 9d e9 60 c3 2b 74 dc  b9
0c19 : bc 5a 65 43 85 89 76 43  c9
0c21 : 9b 68 c3 d2 39 d6 8c 36  24
0c29 : e7 36 29 d9 ce 8a dd 37  d8
0c31 : 2f b9 70 5f 3a a4 ba 7e  f6
0c39 : 73 2b dc 17 8b 4c be e5  3e
0c41 : f3 3c ea 92 f1 69 97 dc  e2
0c49 : be 67 e3 be 97 66 73 6e  e3
0c51 : 49 05 aa 73 ae 49 05 ab  d6
0c59 : 87 66 de 5d 9e 9e e4 82  ee
0c61 : 29 d6 9e e4 82 29 b6 1d  c0
0c69 : 9b 79 76 7a 7b 92 08 ad  75
0c71 : d6 9e e4 82 29 de 1d 9b  55
0c79 : 72 fd 7b e6 7e 3b e2 d6  a0
0c81 : a7 76 4b 69 45 e2 d3 84  27
0c89 : e8 a6 cd cb 5a 9d d9 2d  05
0c91 : a5 17 8b 4e 16 e8 a7 6d  91
0c99 : cb 5a 9d d9 2d a5 17 8b  a7
0ca1 : 4c a6 d0 9d 33 ce a9 2d  d3
0ca9 : 6a 77 64 b6 94 5e 2d 32  14
0cb1 : f2 2e 19 f8 ef 8b c5 a6  df
0cb9 : cf 43 da 2f 16 9b 3f 6b  d8
0cc1 : aa 5e 2d 3d a3 3f e1 ed  25
0cc9 : 17 f2 73 ce ec 14 e8 ee  01
0cd1 : 01 4d 9b 97 f2 73 ce ec  32
0cd9 : 15 ba 3b 80 53 b6 e5 f3  95
0ce1 : fc e9 76 67 71 b9 24 16  fe
0ce9 : a9 df 17 24 82 d5 3b 81  93
0cf1 : c9 20 b5 4e ef af bb 9d  a8
0cf9 : ae c7 24 82 d5 3a d7 92  98
0d01 : 41 6a fd 8a fb f9 58 76  26
0d09 : 6d d9 f2 b6 52 ec f4 f7  46
0d11 : 24 11 56 ba 7b 92 08 ae  74
0d19 : fa fb 9a 57 63 4f 72 41  9f
0d21 : 15 c0 d3 dc 90 45 7c 5a  01
0d29 : 7b 92 08 ae 35 d9 21 7d  67
0d31 : dc 3b 36 f2 ec f4 f7 24  35
0d39 : 11 58 1a 7b 92 08 ae cd  2c
0d41 : f7 34 af 43 4f 72 41 15  5e
0d49 : fe da 7b 92 08 ad be 9e  0c
0d51 : e4 82 29 ce 1d 9b 72 f1  f7
0d59 : 69 96 95 c3 35 77 05 e2  d4
0d61 : d3 2d 2b 86 7d 3b 82 d7  d2
0d69 : b8 67 d3 b8 65 8b ca d9  72
0d71 : b8 02 2a 25 6e 01 0a 58  21
0d79 : b6 2f 36 f1 08 5a d4 ca  cf
0d81 : b5 2f 16 99 5d f5 f7 34  54
0d89 : ae c4 2e 01 78 b4 ca f8  8f
0d91 : a1 71 ae c9 0b ee b7 67  a5
0d99 : cb bc ad 9b 80 22 a2 56  f1
0da1 : e0 10 a5 8b f2 ef 36 f1  cf
0da9 : 08 5a d4 ca c0 2f 16 99  7e
0db1 : 5d 9b ee 69 5e 84 2f f6  79
0db9 : 2f 16 99 5b 78 4e 5b 97  5c
0dc1 : 8b 4e 5b 52 a1 c2 c4 bb  4f
0dc9 : 26 0c 3e 1d 46 9e 96 0c  f4
0dd1 : 3e 1d 46 9e 96 0c 3e 1d  00
0dd9 : 46 9e 1b 56 e5 e2 d3 75  b0
0de1 : e0 3a 2d cd 65 6e ee b5  d4
0de9 : 9d c5 e1 78 b4 ca 87 0b  c6
0df1 : 12 ec 58 30 ca 4a c3 3f  22
0df9 : 33 4b 06 19 54 33 f0 af  79
0e01 : f6 de 32 55 4f 7c 6a 17  4e
0e09 : 64 c1 87 bb 57 45 2d 23  42
0e11 : 45 a2 97 40 98 6d 6e de  02
0e19 : 9d 37 46 19 54 15 3b 2b  38
0e21 : 46 19 49 2a 75 46 8c 3d  c2
0e29 : d9 c9 58 67 e6 1d 43 3f  cd
0e31 : 0a fe b3 c6 4a a9 ef 8d  4d
0e39 : 40 ec 4d 14 95 3a [illegible] 8a  7e
0e41 : 4d d9 9b 4e b6 6e 00 8a  1f
0e49 : 89 5b 80 42 96 2e 6d 3d  f3
0e51 : bc 42 16 b5 39 6d 78 75  36
0e59 : 1a 67 44 e1 d4 69 9c b7  ef
0e61 : 0e a3 4f 0d ac d3 a2 7d  a5
0e69 : 3b 62 a2 1c b6 01 0a 59  50
0e71 : bd d9 30 62 78 d1 2d 84  47
0e79 : df c6 59 bd d9 30 62 15  9c
0e81 : 75 a5 83 13 77 34 e8 98  fa
0e89 : 0f 4a 88 bc a3 ae 9b f8  87
0e91 : cb 1d 74 dc b5 a9 97 ed  86
0e99 : 43 2e 14 32 c9 89 ef 1b  1e
0ea1 : 84 50 ae f2 f0 5b 7a b6  98
0ea9 : a4 56 bf f6 69 42 fb a7  2f
0eb1 : 76 0a ef af b9 a5 76 0d  db
0eb9 : 40 a1 50 6e 5a d4 dd 17  9e
0ec1 : eb df 56 97 ed 43 2e 14  fe
0ec9 : 32 c9 88 5f 3f ce f7 8d  53
0ed1 : c2 28 47 77 d7 dd ce d7  bf
0ed9 : 60 ae c5 de 5f b3 6a 4d  b5
0ee1 : 16 ca d7 fe c3 50 28 54  3a
0ee9 : 1b 96 b5 32 fd a8 65 f1  a2
0ef1 : a1 96 4c 4f 78 dc 22 85  dc
0ef9 : 76 4c 16 db 4d 2c 16 c5  b0
0f01 : 3a c3 6c 6a 05 0a 83 72  19
0f09 : d6 a6 54 33 9b 3a 39 d3  c6
0f11 : a2 fd 7b e3 a1 97 ed 43  22
0f19 : 2f 8d 0c b2 62 17 b3 be  93
0f21 : ad 29 d5 69 4d bd e3 70  39
0f29 : 8a 15 d9 0e 6d a2 ef 48  b2
0f31 : e7 5a 2e cd 40 a1 50 6e  ba
0f39 : 5a d4 cb f6 a1 97 e8 43  d0
0f41 : 2c 98 87 36 d9 3a f7 ba  27
0f49 : 18 58 62 2f bf bc 35 02  c7
0f51 : 85 41 b9 6b 53 74 5b 9a  ce
0f59 : ca d6 5a c3 2d 6a 6e 8b  94
0f61 : f5 ef ab 4b f6 a1 97 e8  4f
0f69 : 43 2c 98 87 36 d9 3a f7  e4
0f71 : ba 18 58 62 2f 71 78 6a  cf
0f79 : 05 0a 83 7e 73 69 0e 9b  26
0f81 : 9c d8 a7 67 3a 2b 75 ce  d1
0f89 : 6d 21 d4 d3 9b 71 9d 94  1b
0f91 : d8 e7 5c 6d d1 4e 9b 9c  59
0f99 : d8 a7 67 3a 2b 75 ce 6d  da
0fa1 : 21 d7 39 b4 87 4c fd 68  36
0fa9 : 66 93 69 bb 36 bc a4 b1  ea
0fb1 : 5d 77 cd af ef a6 c0 e3  32
0fb9 : 3b da 78 1c 6d d6 ef 9c  89
0fc1 : ee 46 eb d6 4a ab 89 86  dd
0fc9 : da 13 a6 fb b2 a1 e9 69  08
0fd1 : 60 cc 2a eb 48 e6 d6 1f  f5
0fd9 : 2c cd 23 9d 7c b3 34 af  fe
0fe1 : e3 b2 d7 c8 c1 99 a4 53  4e
0fe9 : b2 ba 58 33 34 8a dd 15  ae
0ff1 : cc 85 40 bf 36 19 7f 35  1c
0ff9 : f6 ca 85 81 ce f3 fa 47  ed
1001 : 6e b3 f9 8b fd f2 fe b4  15
1009 : 3f 5a ea ab fe db ec a1  6b
1011 : cd 39 b7 bb 30 a6 da 47  12
1019 : 3a f7 66 14 e9 bd 81 ce  9b
1021 : f3 db 14 d8 ed d6 7b a2  eb
1029 : 9d 37 bb 21 db a6 ae 8a  38
1031 : 48 ed d5 7d e7 81 61 9e  e2
1039 : ec a7 7e 06 7e e8 ad d1  e3
1041 : ce ab e5 16 18 90 a8 42  4e
1049 : e9 74 3a 75 5d 19 14 24  e1
1051 : 74 ac 0e a1 9e db 76 77  65
1059 : 4b 3d d5 fe 5c 2a 05 74  ac
1061 : a4 3b 91 ba d5 aa cf 39  c3
1069 : d9 4d 8e dd 14 e8 e4 ab  bc
1071 : ef 2c 0e a1 b5 e9 5b 0a  5a
1079 : 6b 37 78 8d ce 6d da 76  00
1081 : 53 63 9d 76 b7 45 3a 6e  27
1089 : 73 6d ab b2 9b 1c eb 6b  15
1091 : ba 29 d3 73 9b 3b ae 29  e4
1099 : b1 ce 9d d7 14 e9 bd 81  9e
10a1 : a4 cf 6c 53 63 49 9e e8  7f
10a9 : a7 4d ce 6d 61 9e ec e7  e7
10b1 : 59 fb ac d6 b5 ec 9c ee  21
10b9 : c3 3d b1 db ac f7 59 b6  60
10c1 : 15 f4 8e 6d 61 9e ec e7  30
10c9 : 59 fb ac dc 2a f8 e7 3b  67
10d1 : b0 cf 6c 76 eb 3d d6 69  2a
10d9 : 95 f5 27 3a f9 37 55 5e  e5
10e1 : c1 cd be 47 75 5c fb 6a  20
10e9 : f3 9d 7c 9b aa ab 83 9b  8b
10f1 : 7c 8e ea b2 ed af 3e ed  f6
10f9 : f1 b5 c5 36 29 d3 7f bb  a4
1101 : fa cd 21 4d 8a 74 dc b7  03
1109 : f5 99 bf aa bf fc cb 7f  22
1111 : 59 9b eb db 7e aa ff b0  4d
1119 : 5b fa cb 6f f1 aa e5 96  0b
1121 : fe b3 37 fc 2d bf f9 5f  de
1129 : f7 0b 7f 59 9b fe 15 fd  b3
1131 : 42 df d6 5b 7f 7d 57 de  83
1139 : 3b b3 7d dc ed 7a 05 7a  04
1141 : 06 e0 5d 91 9f 83 ce 3b  09
1149 : b2 d7 be 2b d1 3a d5 9f  81
1151 : 43 ed 9d d9 6b df 15 e8  09
1159 : 9d 6a ce fb aa 77 65 af  ba
1161 : 7c 57 a2 75 ab 3a 1e d1  89
1169 : d5 05 fd aa cd 95 45 b4  9d
1171 : 3a ac bc 1c ad 18 6a d5  a5
1179 : 7b f8 39 5a 30 ce 86 56  4a
1181 : 51 d8 85 4d 6f d0 26 1d  9a
1189 : d1 5f 45 f1 2d 9a bb ae  8d
1191 : b0 e3 66 6c b0 97 12 a5  b5
1199 : 43 85 89 61 c4 7a b8 95  5b
11a1 : 2b e1 85 f3 d8 67 d9 78  be
11a9 : 19 ec 2f b6 3a ea aa 99  b4
11b1 : 7e 95 79 a2 2a e8 ef a1  99
11b9 : 9f ed ac 67 ca d9 1d f4  41
11c1 : 6e f4 7e 13 44 ae 8c 33  fe
11c9 : be 12 b3 25 fc f8 78 45  26
11d1 : 43 c3 c4 3b e8 6f 2d a9  a0
11d9 : b2 8a ba d5 d9 d4 7e 7d  73
11e1 : b7 0f 6d f9 2a 3d 6b aa  4a
11e9 : ae 10 a5 b5 d9 ed a6 fd  63
11f1 : 35 eb da 17 f6 ab 25 b5  82
11f9 : bb e7 6c f7 be af 14 9a  b1
1201 : aa d5 7e b3 a1 f0 f2 8e  36
1209 : c4 e1 cd 3b e8 e1 d4 69  5c
1211 : e9 e9 e1 b5 d9 25 6d 0e  b6
1219 : ab 2f 06 86 8c 35 6a bd  46
1221 : fc 1a 1a 30 ce 4a 6f 1a  e8
1229 : 19 50 ef de f8 d8 9b 2e  63
1231 : 65 b4 3a ac bc 1f 83 46  74
1239 : 1a b5 5e fe 0f c1 a3 0d  4d
1241 : bb 34 3f 41 ce aa 9d 4b  5e
1249 : 6a 73 6f ea db 2e f7 87  c4
1251 : b6 6f 2d a9 ce aa 90 d9  78
1259 : ed ac 33 db 5f ff 9c df  0d
1261 : a7 a3 c3 e8 df bd f6 f7  9f
1269 : a7 36 e1 ed 9b 9c ea a9  ff
```

Listing. »Basic-Maker« generiert aus jedem Compilat, das mit den Austro-Compilern erzeugt wurde, ein lauffähiges und editierbares Basic-Programm

```
1271 : a6 c9 b5 b2 8a f8 13 65  47
1279 : b5 fa 7a 3c 3e 8d fb df  d1
1281 : 6f 7b e0 3b ae b0 fa 76  97
1289 : cb bd 34 9c 3d b3 76 5d  d9
1291 : 79 05 fc d7 c7 36 96 d7  ff
1299 : 6a ea db f0 9c eb 6b ba  d9
12a1 : aa f2 76 4e ed b4 d5 f0  e9
12a9 : 26 fd 3d 1e 1f 46 fd ef  dd
12b1 : b7 bd f0 26 f1 9b 2f ab  58
12b9 : fd 5b 65 fd 63 9b 70 f6  40
12c1 : cd ce cc 2b e1 3b 08 af  05
12c9 : 9c ef 3c ac c2 a1 9d 9c  ea
12d1 : 56 11 58 96 07 7c 39 f0  b4
12d9 : ce f9 f3 f1 0d 40 a1 50  d9
12e1 : bf 13 7b 0f 78 dc 22 84  eb
12e9 : 76 66 7c 32 92 3b 0b 3f  a5
12f1 : 10 aa 16 07 43 e2 3d 2a  51
12f9 : 1a ef 7d 6c 4b b2 60 c3  4b
1301 : 2b a5 a5 83 0c ae 65 d9  58
1309 : 3a 82 aa b1 d2 3b a4 55  a9
1311 : d3 36 5d 3c 1e 91 57 50  4b
1319 : bc b7 75 d6 1c 68 65 43  0a
1321 : 5d ef ad 89 2d ab 3c 0e  50
1329 : 9d 86 cf a3 9f 0f 77 b3  29
1331 : de e7 e2 55 6f 6f f6 df  74
1339 : cd 74 bd 61 5e 5c dd 9f  5b
1341 : 47 8c 97 0f a3 b3 de f1  cd
1349 : a8 70 f7 ba ba ba b5 5f  96
1351 : 7c ef 2e da 53 3c ba fb  a6
1359 : 03 b3 38 8f 4a cc 5d ef  96
1361 : ad 85 76 43 ba 5a 39 87  49
1369 : 73 34 b4 73 2c 0e cc cf  98
1371 : ba 2b 31 77 be b6 16 ca  8c
1379 : e9 9f ed e9 e0 c3 d1 cc  f7
1381 : 56 ab df c1 87 a3 98 77  c4
1389 : 33 a8 2b 6a f9 b7 62 c1  93
1391 : e9 71 92 d1 e9 69 60 f4  67
1399 : b8 d4 34 7a 4c f2 eb e5  f0
13a1 : b5 bb e7 6a ec ea 3d 5e  53
13a9 : 29 15 aa fd 78 6d 4e 4b  8a
13b1 : e4 e6 5f bd 3b 99 b4 b6  59
13b9 : b8 c1 a1 fd 3f 05 55 c6  79
13c1 : 96 ad 57 b6 72 5f 27 32  fe
13c9 : da c6 fe 51 a2 6e 6d 73  2a
13d1 : 73 48 dc bf d1 d3 cd cd  26
13d9 : af 30 bf d1 d3 cd 1d 79  dd
13e1 : 85 fe 8e 9d b7 3f 37 95  ba
13e9 : 5e d0 bf d1 d3 b6 e5 66  31
13f1 : e5 57 e2 17 fa 3a 76 d5  24
13f9 : f5 5e 09 7f a3 a7 6c f6  67
1401 : ab c1 34 85 36 29 d3 73  2d
1409 : 6b 8a 6c 53 a6 f2 da ea  82
1411 : ea ea e1 b5 d9 d4 5b 3d  cb
1419 : ef 32 8a 6c 69 0a 74 dc  c4
1421 : d1 17 f6 ab 0e 75 55 40  13
1429 : e6 d7 64 d1 87 2d ae 97  1a
1431 : e4 a8 d3 3a 1f 0e a3 4c  2f
1439 : ec 4e 1d 46 9e 1b 56 e5  44
1441 : fa 55 f6 07 66 67 d9 14  b6
1449 : d8 ec 2c f6 0a 74 de 5b  f8
1451 : 5d 5d 5d 5c 36 bb 3a 82  6f
1459 : 9b 1a 42 9d 37 bb 26 0b  46
1461 : 62 9b 42 74 de c0 bb 6a  68
1469 : e8 4d 8d 21 4e 9b 97 fb  97
1471 : 6b 8a 6c 69 0a 74 dd 97  55
1479 : 5e 41 70 bf 34 2f 2d 58  ae
1481 : 5e ed df 3a 17 cf 77 24  2b
1489 : ba df 8c de 71 7f 6a b2  54
1491 : 5b 53 79 dc 3a 8d 33 a1  b0
1499 : f0 ea 34 ce c4 e1 d4 69  67
14a1 : e1 b5 bb e7 57 ed 7d 9a  59
14a9 : e2 68 c3 56 bf f6 37 96  39
14b1 : d7 57 57 57 0d ae ce a3  bd
14b9 : a9 ce aa ea ec f2 8a 1e  9e
14c1 : ce 69 5c 76 7f 49 b2 da  ec
14c9 : ea ea ea e1 b5 d9 e5 14  09
14d1 : 3d 9c d2 b8 fb 3a 86 57  82
14d9 : fd b7 64 2f fa f7 37 75  e8
14e1 : da 5e cd 71 2a bf b3 c0  7d
14e9 : 2f ed 56 6e e5 b5 fc 95  9d
14f1 : 1a 67 43 e1 d4 69 9d 89  ee
14f9 : c3 a8 d3 c3 6a cf f4 e8  48
1501 : 1c d8 bf ee ae bf 97 d4  48
1509 : b6 aa f0 61 67 61 f9 ec  c0
1511 : b5 db 33 97 f8 0b fa b5  b3
1519 : cb fa ed cb fd 35 c7 79  f2
1521 : e5 66 1d 9c 56 15 dd 8f  b9
1529 : 43 2f 06 67 0f 8a ae ad  ce
1531 : 57 ec b0 3b a5 9e 91 57  d6
1539 : 47 73 33 e8 6d 34 cd c2  59
```

```
1541 : f7 8a 11 a4 dd db 78 3a  69
1549 : 29 69 55 6c 78 d4 35 6a  e2
1551 : bd bd 95 d5 b6 d5 5d 14  c5
1559 : aa b6 0d 40 a1 50 e8 13  10
1561 : 0d a9 be 91 56 7d 08 26  e2
1569 : 93 92 49 5a 75 5f d8 cf  38
1571 : de 9b 2d b6 1e d9 99 bf  d6
1579 : 81 9f b7 b6 ad 9b 80 22  8c
1581 : a2 56 e0 10 a5 8b fb 7b  26
1589 : 6d bc 42 17 8b 4c a8 70  66
1591 : b1 2e c9 34 e8 98 0f 74  cb
1599 : 61 e9 1c b6 01 34 61 97  33
15a1 : 8b 4c a8 90 96 6f d0 d8  68
15a9 : fc ac f9 75 30 be 73 7c  e8
15b1 : eb bf 64 b4 ae 19 e2 6d  46
15b9 : 8b a7 e7 16 bd c1 78 b4  0a
15c1 : cb e3 ef 70 ec dd 78 0e  43
15c9 : 8b ee 5c 17 fe 9e d1 7d  ec
15d1 : cb 8a dd dd 6c bb 39 bc  17
15d9 : 92 0b 57 c0 e4 90 5a a7  6a
15e1 : 44 e4 90 5a a7 2d c9 20  52
15e9 : b5 4d 27 b9 c9 20 b5 70  9b
15f1 : ec db 97 8b 4c be 3e f7  c6
15f9 : 0e cd d7 80 e8 bf f2 be  c9
1601 : ad dd d6 d8 4b b3 66 fb  51
1609 : a2 cf f5 ed 8d f3 ae fd  fd
1611 : 92 d2 b8 67 81 d1 65 3e  e0
1619 : 8d 6c dc 01 15 12 b7 00  f4
1621 : 85 2c 5a 7d 1d bc 42 33  2a
1629 : c4 db 56 cd c0 11 51 2b  5a
1631 : 70 08 52 c5 f1 36 db 78  24
1639 : 84 67 91 d1 ad 9b 80 22  0d
1641 : a2 56 e0 10 a5 8b e4 74  7c
1649 : 76 f1 08 cf 79 d1 ad 9b  c8
1651 : 80 22 a2 56 e0 10 a5 8b  92
1659 : ef 3a 3b 78 84 bb 23 3b  6c
1661 : 2e 89 78 b4 cb e3 ef 7f  a3
1669 : fc b3 aa 94 ce 6f 8f 87  32
1671 : 66 54 38 58 9b 2b aa a7  27
1679 : fb 2c 42 d2 b8 3a ce c3  95
1681 : 89 70 55 99 7f e3 70 77  93
1689 : 64 ab a2 d9 ac 3a 1e cb  53
1691 : 10 be 05 c1 7f e3 70 5b  09
1699 : fa cb 6d ab 5b [illegible] [illegible] 37  1a
16a1 : 3a ce c3 89 59 b4 c3 b3  16
16a9 : dc bd 11 4b 6e 65 08 a8  95
16b1 : 8d ee c9 83 0e 6e 00 8a  02
16b9 : 6d a3 0f 4b 06 1e 01 0a  8f
16c1 : 75 a3 0f 72 41 58 67 b6  fc
16c9 : dc d7 13 3d d6 a3 d7 7c  e0
16d1 : da f1 37 3a ce c3 89 e0  ac
16d9 : b3 fa 7a 3d 02 61 dd 4b  89
16e1 : b3 29 28 54 3e c7 a5 7f  75
16e9 : e0 cb b3 39 2d 95 06 fb  52
16f1 : 91 08 ab a8 13 76 57 59  7b
16f9 : a6 55 4a c0 ac 79 81 e0  53
1701 : 95 0c ea 16 d4 fa f9 62  eb
1709 : aa af 39 2e be d4 55 57  36
1711 : de 76 58 b7 3b 51 4b 86  b0
1719 : de 5c 36 6c b7 b8 12 b6  37
1721 : 54 3e ad 75 57 d4 da 37  a4
1729 : 96 db 0f 6c cb 4f c0 5f  f7
1731 : ff 95 dd 42 5b 62 14 d8  85
1739 : d2 14 e8 b9 da 1d 09 58  f2
1741 : 6d 5b 95 50 56 51 53 6d  e3
1749 : b8 42 97 7b b2 df e1 fa  1f
1751 : 90 bc c3 6b 8a e9 b7 67  43
1759 : d9 de d6 cd c0 11 51 2b  41
1761 : 70 08 52 c5 fb 3b dd bc  ad
1769 : 42 1d 10 ad e1 cb 14 1d  fb
1771 : 0a eb b1 61 fa 90 bc cd  cc
1779 : 3c 3e 99 af 2d eb 84 5e  32
1781 : 5d 32 ee 32 8b f7 dc 7f  e4
1789 : c9 1e 97 8b 4f ec 12 ab  b5
1791 : 2c a8 90 96 6f 2d b6 1e  80
1799 : d8 b6 9f 80 bf 7d c1 af  12
17a1 : 2d eb 85 d0 ae bb 16 1f  9f
17a9 : a9 0b cc 34 45 74 db b3  60
17b1 : ca a8 ad 9b 80 22 a2 56  fe
17b9 : e0 10 a5 8b e5 54 6d e2  f9
17c1 : 10 be fc 9d 42 55 4a 2f  79
17c9 : 6f 50 5f a6 42 f8 42 f3  ea
17d1 : 8b ef c9 d4 25 54 a6 7e  ed
17d9 : 5a 8e a3 cb 6f 7c ee 9d  ae
17e1 : 7c a6 6c 7f 35 b4 a6 7b  46
17e9 : e7 06 fc 05 0e ee a2 17  c4
17f1 : 1c de 99 72 5c 1d bc 2a  27
17f9 : 21 c0 52 cc 9d bd a5 45  91
1801 : ae 3d a1 78 b4 6d 35 8b  68
1809 : 8a 8d ed 14 61 9d d3 af  09
```

```
1811 : da f4 21 b3 9d e3 9d b6  01
1819 : aa 94 5b 57 06 d7 15 ea  18
1821 : 1a 22 bc c6 7d f7 0c ff  1c
1829 : e6 a2 b6 6e 00 8a 89 5b  0d
1831 : 80 42 96 2f ff 35 1b 78  65
1839 : 84 2f 6b 50 cd 5d c7 42  a5
1841 : 56 1b 52 fb f2 75 09 55  e2
1849 : 28 bd bd 41 7e 99 25 fa  27
1851 : 98 79 52 fc cc 39 b7 64  18
1859 : 85 93 d4 1d 55 7f 8d 94  31
1861 : 5f a6 4e a1 2d b9 5d 41  74
1869 : db 68 69 55 7f 9f 50 96  e1
1871 : d8 9d 47 96 d7 05 fa 64  17
1879 : d2 67 ff 95 05 fa 64 85  76
1881 : 50 6f e1 2b 24 ee 9e 6c  74
1889 : a2 dc d6 1d d3 af b1 3b  eb
1891 : d4 2a f4 ef 30 ad fb 72  fb
1899 : f9 d5 25 8b 6c 5d c6 53  ab
18a1 : 33 9c 74 09 86 d4 ec a2  69
18a9 : 86 74 d2 b8 e7 54 33 25  6d
18b1 : c5 84 99 b6 36 05 db 57  9f
18b9 : 42 be 96 d9 3b fe 74 0c  d1
18c1 : fb eb fd ae d3 4f 56 a8  69
18c9 : 4d cb 9c e2 ab f5 d8 17  7b
18d1 : 6d 5d e0 58 71 2e 17 ff  15
18d9 : 47 75 d8 02 aa cf 67 cb  b0
18e1 : ca ad 9b 80 22 a2 56 e0  cb
18e9 : 10 a5 8b f2 f2 b6 f1 08  ca
18f1 : 5e 2d 39 76 65 5d 1d 13  df
18f9 : 92 41 6a 9c b7 24 82 d5  ac
1901 : 36 bb 92 41 6a e1 d9 9d  3a
1909 : 74 54 48 4b 37 65 f6 55  48
1911 : 6d 69 6f b2 b6 f1 09 77  73
1919 : 5c 5f 1f 7a 75 d6 6c a6  49
1921 : 7e c6 ef 0e cf 72 41 15  80
1929 : 13 73 5c 42 96 6e 5e 2d  06
1931 : 32 a2 42 59 b9 cd b6 ae  b2
1939 : aa a5 33 b2 ca 67 16 99  4c
1941 : 7e 95 79 75 d9 47 7b 25  a7
1949 : 5d 1a 49 7b ed 93 c2 ff  fc
1951 : aa b9 7d a9 d7 55 f9 6d  57
1959 : d9 63 6d 87 65 09 8a da  af
1961 : df 1f 27 ba 53 bf 1e 17  cb
1969 : 74 ad d5 dc ac 17 65 36  4a
1971 : d2 c1 76 53 ad bb 62 17  9c
1979 : 5d 95 2e cb 64 c1 bb e6  b7
1981 : 75 eb 8e a8 97 95 b2 9a  cb
1989 : 5e e6 bb a0 4c 36 a5 fc  64
1991 : d7 d2 dd ec 9b 1c d9 9e  25
1999 : ad 71 7e bd f1 d5 12 f2  52
19a1 : b6 53 4b dc d7 74 09 86  c2
19a9 : d4 bf 9a f8 e6 c5 7c 25  fb
19b1 : fa f7 c7 36 96 ef 65 f0  c0
19b9 : 97 d5 ae 2f ea d7 2e 26  3f
19c1 : ec b1 b6 66 84 da d9 b8  f8
19c9 : 02 2a 25 6e 01 0a 58 ba  2f
19d1 : 13 76 f1 08 5c 29 bf 54  53
19d9 : 96 d8 ac ee 32 99 fa d3  68
19e1 : 6b 66 e0 08 a8 95 b8 04  db
19e9 : 29 62 fe b4 dd bc 42 17  94
19f1 : 0a 6f d5 25 b2 65 9d 36  06
19f9 : 03 36 b9 45 ad 4c ab af  7a
1a01 : 91 b2 f2 ae c8 dc e7 55  3b
1a09 : 5e f9 cd ad bf f1 71 bb  56
1a11 : 55 5c 18 9b 4d 2c 18 9b  dc
1a19 : b8 0d ce ba b8 e3 44 2a  73
1a21 : 21 cb 60 10 a5 9b 97 f3  bf
1a29 : 5f 6c 9d 5b 4a 65 d7 90  e1
1a31 : 73 6d 17 6d ee ef b0 66  cc
1a39 : 15 7b a5 83 30 ad fb 76  33
1a41 : 79 73 6b 66 e0 08 a8 95  38
1a49 : b8 04 29 62 f9 73 76 f1  93
1a51 : 08 5e 2d 32 a1 c2 c4 97  8c
1a59 : 66 74 4e 49 05 aa 72 dc  df
1a61 : 92 0b 57 0e ce ec 98 30  28
1a69 : f4 8a 89 83 0e db 43 48  d2
1a71 : a5 8e 87 37 01 ee d0 ec  cb
1a79 : 4c 02 6e e0 33 c0 b8 67  69
1a81 : fe 7f 93 36 5c aa d9 b8  de
1a89 : 02 2a 25 6e 01 0a 58 bb  f1
1a91 : 2e 56 de 21 25 d9 e9 ee  6d
1a99 : 48 22 96 d3 dc 90 45 44  02
1aa1 : c3 b3 6e cf 83 f8 0e 6c  e4
1aa9 : 57 44 e7 45 6f 59 cd fc  b8
1ab1 : 05 fc d7 c7 3a da ee ad  b5
1ab9 : a5 32 eb c8 39 b1 5b a3  61
1ac1 : 9d 94 ec bf 5e f8 e6 c5  b0
1ac9 : 74 4e 74 56 f5 9c 6f b6  bc
1ad1 : 5a d4 e6 95 75 62 5f 8b  01
1ad9 : ca dd 9d 75 37 02 f7 d5  b7
```

```
1ae1 : fa af 74 8d 22 76 3e 87  5f
1ae9 : ab f5 5e d8 78 18 1f 56  b3
1af1 : ed ea 6b ca 67 81 71 2d  2a
1af9 : ad 56 86 1c d7 56 07 66  10
1b01 : 67 d9 1d 85 9e c5 54 a2  fb
1b09 : fd 2a fa dd d9 d5 1e a9  ae
1b11 : 1e 9d 95 ea 89 e9 d3 7d  f3
1b19 : 57 af 7a 04 c3 6a dc b5  d5
1b21 : a9 d8 33 fe fc a2 d6 a6  70
1b29 : e8 be 75 4d 6b 3e ef 28  30
1b31 : af 86 e2 e2 e2 c3 3f e1  45
1b39 : 6e cd b7 2a b6 6e 00 8a  b5
1b41 : 89 5b 80 42 96 2e db 95  56
1b49 : b7 88 42 d6 a6 55 d4 d3  c0
1b51 : a2 15 ea 71 ae 8a 88 72  ad
1b59 : c5 79 98 04 29 6f 1e c6  96
1b61 : 55 bf 96 a7 8f 7d 2a df  7e
1b69 : e8 50 d6 f6 f6 32 8d 52  ea
1b71 : de fa 51 d3 1d 1d 9a e8  92
1b79 : da e2 a6 1d 64 e8 e6 1d  75
1b81 : 1b f5 4a b2 35 e9 4c 15  7e
1b89 : 9b 60 76 63 ae 23 d2 b3  15
1b91 : 0e c2 75 c4 21 58 47 67  b7
1b99 : 3a ea e1 d2 34 85 67 15  52
1ba1 : 94 54 d3 79 45 54 4b b3  0f
1ba9 : fc a5 ba ee b3 63 f9 a5  8e
1bb1 : d2 fc be 3f 96 a5 bd 8c  3f
1bb9 : af 1f e8 52 de fa 55 69  6a
1bc1 : 59 d5 a5 61 56 95 99 5a  c8
1bc9 : 53 15 a5 59 56 95 9b 5a  71
1bd1 : 54 c3 48 53 56 e5 ef bb  4f
1bd9 : ac b7 52 b6 c3 88 f4 a5  6c
1be1 : eb 78 84 2b 04 d7 36 ff  67
1be9 : d9 5c 6f 76 df 62 b8 e1  53
1bf1 : 5b fd 7e ef 53 c2 aa d0  80
1bf9 : ad e2 6d 6d ec 65 56 f1  58
1c01 : 09 6f 7d 28 bd f7 75 9e  d5
1c09 : a2 86 e3 db fe c7 a6 f8  1d
1c11 : b6 fe fb d3 97 2a 61 d8  c2
1c19 : 25 66 dd ca c1 98 54 4d  0f
1c21 : 2c 19 9a 45 2d 83 30 af  38
1c29 : 3e df 25 ee 96 0c c2 b3  ba
1c31 : ad e1 3d e8 7c df 94 bd  d0
1c39 : fe d8 bf bb cf db c4 25  44
1c41 : 6d 6e de 21 2b 66 e0 08  3b
1c49 : a8 95 b8 04 29 66 71 39  68
1c51 : e5 f8 9a 17 64 c1 89 e3  7e
1c59 : 44 aa 7b e3 2d 71 b4 5e  3c
1c61 : b3 73 26 c6 de 9f 3f 73  ff
1c69 : da b1 b7 87 cf 66 f3 9f  3a
1c71 : b9 93 7d 6f 4f 9f b9 ed  f6
1c79 : 5f 5b c3 e7 b3 79 cf 66  87
1c81 : 2f 3e b6 6e 00 8a 89 5b  7c
1c89 : 80 42 96 e8 10 bc 5a 65  08
1c91 : d0 e7 ed e2 10 bc 5a 7d  78
1c99 : 01 5b ee 5e fd af 8d ef  43
1ca1 : 5b ea 3d f0 21 ee 5e db  99
1ca9 : 5f 1b e6 5b 92 51 bc fa  57
1cb1 : af 04 df 2e dc 92 8d a9  0c
1cb9 : b7 14 a6 fd 6f 39 ef 43  eb
1cc1 : e6 fc a6 fa 45 59 f8 fb  29
1cc9 : 97 a5 44 f1 fe 37 a5 2c  1b
1cd1 : 5e cd c7 8f f5 1e d5 4a  36
1cd9 : db c4 23 38 9c f3 bb 2d  99
1ce1 : 7b e2 bd 16 62 fd b3 bb  5c
1ce9 : 2d 7b e2 bd 13 bb 0c f9  77
1cf1 : 9f 50 df bd 76 f5 9e fe  f7
1cf9 : f0 bf 8f 9e cd 5d c1 bf  cf
1d01 : 7a ed eb 29 5c 17 cf f3  37
1d09 : 8e ef ab e5 17 ad ec 9d  04
1d11 : 6b d4 fc 4b d8 9a 42 bb  72
1d19 : 05 70 2d 93 ea 46 5f 43  f9
1d21 : 02 c4 ab 53 bb 05 70 2e  dd
1d29 : c8 77 7d 5e f6 ee ba 17  d8
1d31 : 60 db fb 0c fb 5d dd b5  8d
1d39 : c1 dd f7 b9 df 76 3b ee  9a
1d41 : 02 b5 5e db 73 48 5f 13  4e
1d49 : ce 2b ee 95 df 1c ea be  23
1d51 : b0 de 69 5d f5 81 a4 cf  14
1d59 : 6c 53 63 49 9e ea da b0  7f
1d61 : ae c1 cd 8a e0 1b 7e 55  40
1d69 : a9 dd 86 5f d4 9a a9 56  04
1d71 : a7 36 2b b1 6d 4e bc 4d  0b
1d79 : fd 6b 5e 77 02 47 62 bf  16
1d81 : f6 37 2a d5 bf 8c da a9  77
1d89 : ef 8c e9 b9 cd bd 66 dd  90
1d91 : 42 55 3d f5 9d 37 c3 ba  a4
1d99 : 34 85 65 15 34 d7 d6 e4  b3
1da1 : 94 6f ff 6d c5 28 be 94  5c
1da9 : b9 77 5b 72 4a 2f 7d e7  27
1db1 : 5b 61 57 ca 65 d7 90 5f  02
1db9 : f3 fb 47 5a d8 71 3f 15  07
1dc1 : 8e ea 6e 00 93 bf 7b 81  88
1dc9 : dc 2a d4 dc b2 aa cb d5  e6
1dd1 : f6 8d f8 6e c8 5d 67 b4  18
1dd9 : 6a 57 64 2e af ed fa d6  d2
1de1 : be b5 5d 57 11 be 0c 4d  8e
1de9 : a6 e4 92 8a 6d b9 14 a2  31
1df1 : 9d 78 d1 2d bb 4d f0 62  93
1df9 : 6d 37 24 94 53 bd c8 a5  2f
1e01 : 15 ba f1 a2 5b 6d 9b e0  95
1e09 : c4 da 1c db 92 49 47 3a  c2
1e11 : e4 8a 57 8d 12 d8 4d fc  d9
1e19 : 65 9b 9c db 6a ea da 53  fe
1e21 : 3b 2c a6 71 69 96 b5 3d  e7
1e29 : dc b6 be c9 25 70 ea 34  33
1e31 : fd 91 4a e1 d4 69 fb 2f  ac
1e39 : 65 70 ea 34 f0 da b7 2d  37
1e41 : 6a 6e 8b 73 59 5b 77 29  d4
1e49 : 9d c5 e1 6b 53 bb 94 ce  b2
1e51 : e2 f2 c1 9f 33 f3 78 ff  c5
1e59 : 5f bb d4 f0 ad 84 dc d4  05
1e61 : 0a 17 43 de d9 d6 70 f4  a3
1e69 : 35 6a bd fb bf 52 17 87  bc
1e71 : a5 0b de 6f e3 fd 7e ef  49
1e79 : 53 c2 b6 13 72 85 d0 54  7d
1e81 : da 82 aa 36 6f b8 6a e9  48
1e89 : d5 7b f7 7d b8 5e ee 94  2d
1e91 : 2f a5 b9 78 f2 d9 fd db  be
1e99 : 67 46 f7 5d 1b f6 5d 58  5c
1ea1 : 9a 81 42 db ff 65 71 b8  6a
1ea9 : 45 0a b6 cd 02 bb a5 fe  ed
1eb1 : 35 2d 32 14 0f ec ae fe  9d
1eb9 : 8e ef db fb 02 ef 2b 3e  7e
1ec1 : e7 df 17 fe 05 fd ff 7c  77
1ec9 : 59 64 e4 fd f1 57 93 f7  65
1ed1 : fd f1 57 91 ef df 14 72  02
1ed9 : 2f f7 05 1c 8e be e0 b0  8c
1ee1 : 89 4b ee 0a 51 3f 76 b8  67
1ee9 : b0 89 bb fd c2 e3 13 ff  a4
1ef1 : df fb 17 18 8f 1a cf c7  2f
1ef9 : 79 8b 69 45 e4 f6 b4 11  36
1f01 : 79 8f 7a 8d eb c7 73 f5  49
1f09 : 3b d4 78 ef 1d cf d4 ef  4e
1f11 : 51 79 8e ef 59 e5 16 b7  4d
1f19 : b8 ef 19 6a 8b c6 d6 88  b8
1f21 : da 3c a3 de cf 78 ef 15  89
1f29 : e5 16 b1 51 b4 a2 d5 e3  2f
1f31 : 6b 48 bd ee fd e5 16 a8  c6
1f39 : f7 b4 67 bc a2 d7 59 e4  14
1f41 : 54 6d 35 1e 5a 4f 78 95  8a
1f49 : eb b4 5e 33 69 7b 8e f2  1f
1f51 : 2d ef 15 ab ce 2b 5a 2e  3d
1f59 : f1 7a 4f 31 da e2 bc d4  e3
1f61 : b4 ba 6a f3 52 8b b9 e8  c6
1f69 : f1 de 6a 51 77 3d 17 97  7b
1f71 : 36 93 d1 78 cb b8 a8 bc  93
1f79 : b9 66 8b cb 9d 6c 77 91  00
1f81 : 7a 51 5e 5a 6a 35 9e f2  37
1f89 : e5 9b 48 ad 5e 36 6a 8b  5c
1f91 : c9 ed 52 78 8d ee fd 37  ab
1f99 : 88 da 3c c5 9e f2 2e a5  dc
1fa1 : ca 6f 11 74 d6 7b c9 f6  54
1fa9 : 88 bc 8a 8b 5d 47 9b e7  f2
1fb1 : 96 4f 3d c7 93 1e 79 6f  26
1fb9 : 2f 67 e3 bc b4 a2 f3 19  8f
1fc1 : e3 f7 90 5e 45 6d 3d e3  0c
1fc9 : b9 e8 bc bd df c5 79 de  ad
1fd1 : c5 a2 f3 16 8b 57 9a 96  b2
1fd9 : 91 5e 45 3e 8b ca 33 f1  72
1fe1 : de 33 68 d9 e3 5a 1a 8f  47
1fe9 : 2e 6d 22 bc 8a ee 7b c4  86
1ff1 : 6f 67 bc bd 46 7b cd 46  ff
1ff9 : ad 6e 9e 32 d6 7b c8 a8  89
2001 : d1 92 f3 8b 7a cb cb d8  70
2009 : b7 2f 2e 5d 51 92 f1 96  2e
2011 : b8 a8 c9 79 45 db 67 48  20
2019 : c9 79 ac ef 1e 4b c6 d6  cd
2021 : 9c 2c 27 95 c4 21 0b ff  d1
2029 : 55 03 9a 34 04 d3 26 16  d1
2031 : 41 75 4a 34 26 8e 47 b1  9d
2039 : 54 a3 42 68 8c 89 fb 60  c2
2041 : d0 5b 8c 66 82 dc 59 a1  66
2049 : 97 ba bd 2c b7 1b f6 9b  9a
2051 : 5d c3 51 91 3f 94 33 af  db
2059 : f8 59 64 4f c9 06 84 51  82
2061 : 02 63 6f 5a 09 44 05 27  51
2069 : 1b 7a d1 ea 88 09 3c 6d  30
2071 : eb 49 4a 20 2e d6 36 f5  f6
2079 : a1 37 0d 86 44 ff 48 34  98
2081 : 22 88 6d 6e 9a bb 9e d7  c2
2089 : 1b 31 a0 94 42 9f 9f 65  62
2091 : bf e9 22 d6 c9 1b 4b a6  98
2099 : b3 f1 b3 1a 3d 51 0a 79  ee
20a1 : d6 5b fe 92 2d 6c 91 b4  1d
20a9 : ba 6b 3f 1b 31 07 9f f1  fa
20b1 : d3 d0 e1 73 8b df de c5  92
20b9 : 77 73 65 ad 7b 74 d6 89  c3
20c1 : 74 5a dc da 6b 6a 3b 65  b6
20c9 : ad 12 cb e2 f9 ec 67 70  54
20d1 : b9 c5 e8 64 c2 84 82 65  59
20d9 : 96 d1 a3 42 f8 59 6d 13  bf
20e1 : 4a 39 7e 7c 4b 0f 08 bc  be
20e9 : 5a 37 b3 ee 8b a2 ed ae  8c
20f1 : b3 59 f1 6e 5d 4f 6a ef  75
20f9 : 1d ad 1c 98 50 90 e5 68  f9
2101 : 79 fc 12 e8 b5 b9 b4 d6  44
2109 : d4 76 cb 52 ef 7b d8 a8  e5
2111 : d1 b4 2e e6 d3 5b 53 26  56
2119 : 14 24 14 d0 ce e0 97 45  3b
2121 : ad cd a6 b6 a3 b6 5a 97  be
2129 : 7b de c5 46 8d a1 71 75  e4
2131 : 1a b5 c7 c9 85 09 05 34  6e
2139 : 33 38 25 d1 6b 73 69 ad  5f
2141 : a8 ed 96 a5 de f7 b1 51  51
2149 : a3 68 5e fd 97 68 f3 32  69
2151 : 61 42 43 bb a1 85 c1 2e  c5
2159 : 2a 37 b4 51 e6 35 76 cb  00
2161 : 59 91 6e 5d 51 76 8a 29  0f
2169 : 35 76 8b 55 8b c9 3b 1b  91
2171 : f1 42 84 83 4d 0f 2b 82  14
2179 : 5e fd db 38 f1 6e 5d 3b  52
2181 : a3 ad 7b de c5 6d 7b bf  eb
2189 : 6a 5e a5 ec 58 a5 e7 2c  d4
2191 : 5e 49 bb fc 98 50 90 ee  4e
2199 : e8 78 5c 12 f5 af 68 30  f6
21a1 : 5b 37 ad 67 b6 6a 5e 3d  a3
21a9 : ea 37 b6 5c c6 ae d9 6b  88
21b1 : 3d 62 f2 4e 6b f8 a1 42  2f
21b9 : 43 95 a0 87 06 66 85 87  99
21c1 : 06 66 84 4e 09 6d 68 a4  cc
21c9 : d5 da 2d 68 f7 b3 db 2c  49
21d1 : 5e 49 cd a0 e6 64 c2 85  03
21d9 : ea 38 76 6b bd b9 06 29  fe
21e1 : ea 1a ef 6e 41 df 6b 54  0b
21e9 : e3 ee fc 2f 60 d7 68 0b  e5
21f1 : 09 ba 92 f1 9a 5c 3a a9  03
21f9 : b9 7d df 7b 07 63 7f 1d  9c
2201 : c6 34 0e 26 41 ae d0 16  23
2209 : 13 75 25 e3 34 b8 ba a9  e4
2211 : b9 7d e2 7b 0d 05 b8 c6  1a
2219 : 81 c4 c8 35 da 02 f2 9b  96
2221 : a9 2e cd a0 ae aa 6e 5f  21
2229 : 6f 3d 86 84 dc 59 a1 97  b7
2231 : 3d ab b9 ee 37 f0 38 99  9f
2239 : 06 bb 40 5e 53 75 25 d9  22
2241 : b4 25 d5 4d cb ef 8f d8  d3
2249 : 68 2d c5 9a 19 7b ab d2  ce
2251 : cb 71 bf 81 c4 c8 35 da  12
2259 : 02 f0 9b a9 2d ab 4c 57  ff
2261 : 55 37 2f be e7 b1 2d ab  0e
2269 : 4d ad 2d c3 58 1c 4c 88  f9
2271 : 0a 59 9c d7 f8 ee 1a 9a  ee
2279 : ed 01 67 37 50 d7 68 0b  23
2281 : 31 bd 4e 3b ed a8 c7 29  21
2289 : 3e 58 76 b8 39 12 3c 90  de
2291 : 72 9f 5c 20 79 03 18 93  25
2299 : e5 87 6b 27 2a 47 92 0e  45
22a1 : 53 eb 84 0f 20 68 6f ba  65
22a9 : 7a 18 b7 a8 de b2 5d cd  c7
22b1 : a6 b6 a3 b6 5a 96 d6 8d  43
22b9 : 1b 4a 2c 16 d6 7e 39 76  fa
22c1 : 98 a5 d1 6b 73 69 ad a8  98
22c9 : ed 97 73 db 31 c1 c9 84  2b
22d1 : d1 ef a2 d2 57 a2 d2 b3  da
22d9 : d1 43 9d 45 ad ee 3f 49  3e
22e1 : 8c 98 4d 3d 65 1a 13 45  b2
22e9 : a7 ac a3 4a e5 1a 13 cd  30
22f1 : f5 1c 3b 35 da 02 9e dd  5e
22f9 : f7 d9 1e 75 ff d9 e5 90  9b
2301 : 91 ff b7 ee c0 ef 72 10  d3
2309 : df 74 f4 22 f4 a2 a2 d7  42
2311 : 58 b6 b4 68 da 51 60 b6  26
2319 : b3 f1 cb b4 c5 2e 8b 5b  01
2321 : 9b 4d 6d 47 6c bb 9e d9  7a
2329 : 8e 17 82 5f 07 26 14 0c  d9
2331 : 51 c2 85 f6 69 ea 51 b4  c0
2339 : 6d 46 f7 5b 5a 7d ee b3  e7
2341 : 5f e7 1f 35 c3 2d 2b 3d  cf
2349 : 16 95 8a 34 b1 51 a5 62  54
```

Listing. »Basic-Maker« (Fortsetzung)

```
2351 : 88 3d 77 9b d6 57 68 c3   1a
2359 : 47 be 8b 47 aa 34 b1 51   81
2361 : a3 d5 0d 65 0c b7 9a 8f   e7
2369 : 28 a5 d6 57 68 c3 49 5e   8b
2371 : 8b 49 4a 34 b1 51 a4 a5   3d
2379 : 10 cb b3 5b 7f 64 6b 29   e2
2381 : 21 96 f3 51 e2 49 75 95   8e
2389 : da 31 09 0b 1d 0d 4b d8   b9
2391 : b1 4b f3 f8 39 30 95 2e   cc
2399 : 61 9e 28 0c 4e 53 d7 05   be
23a1 : 6d 66 2a 5c c9 1e 28 27   d4
23a9 : e9 02 a5 cc 91 e2 82 7d   cb
23b1 : 40 54 b9 92 3c 50 4f f9   53
23b9 : 81 52 e6 48 f1 41 3f a8   9d
23c1 : 0a 97 32 47 8a 09 f5 41   58
23c9 : 52 e6 48 f1 41 3d 98 48   d0
23d1 : fa 80 9f 5c 15 b5 98 62   a5
23d9 : e0 3a 83 df 79 9c 78 18   42
23e1 : 3c 73 17 01 fe 67 29 b8   fe
23e9 : 03 35 c0 7c e8 a9 f5 c1   7e
23f1 : 5b 59 8a ec 30 62 e0 3a   47
23f9 : 83 df 07 1e 06 0f 1d 5d   fa
2401 : 86 0c 5c 07 5f 47 77 94   bd
2409 : 35 79 73 20 5f f1 c6 44   05
2411 : e5 81 9b 30 18 9c a7 d7   58
2419 : 05 6d 66 2b b0 c1 8b 80   1c
2421 : eb 35 94 31 70 19 73 ea   66
2429 : 6e 54 31 70 1a 07 d4 e3   11
2431 : a8 62 e0 35 b5 94 31 70   8f
2439 : 1f 36 7d 4d cd 4e 3a 86   c2
2441 : 2e 03 fa 9a ca 18 b8 0f   71
2449 : f4 cf a9 b9 a9 c7 6a fa   3f
2451 : c6 4c 03 a4 99 cb 01 88   a0
2459 : f7 d3 e4 98 b8 0e 09 ea   7c
2461 : 18 b8 0f bd 3e a7 1d 43   6d
2469 : 17 01 a4 7d 4e 3c ce f0   bd
2471 : 2e 54 31 70 1e 29 f5 38   97
2479 : ea 18 b8 0d d1 f5 38 ea   c3
2481 : 18 b8 0f 50 fa 9c 77 d3   dd
2489 : e4 c0 3a 49 9c b0 18 a8   86
2491 : be 7d 24 c5 c0 70 4f 50   3d
2499 : c5 c0 7d e9 f5 38 ea 18   58
24a1 : b8 0d 23 ea 71 e6 77 81   35
24a9 : 72 a1 8b 80 dd 1f 53 8e   20
24b1 : a1 8b 80 f5 0f a9 c7 7c   4d
24b9 : fa 4c 03 a4 b1 2c 14 63   c2
24c1 : 5c 27 5e 19 2c 06 28 a9   53
24c9 : cb 01 91 3e f0 20 7e 19   7d
24d1 : 26 2e 03 25 15 25 b6 68   9a
24d9 : 4b aa 9c 7d df e1 92 c4   31
24e1 : b0 51 8d 70 9d 78 64 b0   3c
24e9 : 18 a2 a7 2c 06 44 fb c0   b6
24f1 : 68 4d c5 a2 bb 0c 1a 7a   3f
24f9 : 02 5a d7 77 fd 32 65 da   ca
2501 : 34 25 d3 ef 26 4b 41 6e   59
2509 : 2d 20 7f 4c 98 1f 7b 8e   3d
2511 : 62 e0 3a f3 df 7c d2 55   c8
2519 : e5 cc 81 b3 90 62 e0 39   4d
2521 : a7 be a8 ed 2b f8 b3 4d   f3
2529 : 3d 01 2c dd f5 47 6a d6   fd
2531 : ce 41 8b 80 e6 9e fa 67   b1
2539 : 68 cf 48 06 27 26 6b 84   b6
2541 : f7 d3 3b 50 3e 4e 39 8b   4d
2549 : 80 e6 9f bb d9 c8 31 70   25
2551 : 1f 28 f5 0c 5c 06 a1 f5   ec
2559 : 38 ea 18 b8 0e c0 fa 9c   30
2561 : 75 0c 5c 07 b8 3e a7 1d   2b
2569 : f7 63 b4 ad ec c8 1b 39   e8
2571 : 06 2e 03 e7 9e a1 8b 80   72
2579 : ea 4f a9 c7 50 c5 c0 79   97
2581 : 07 d4 e3 be fc 9d a5 76   03
2589 : 18 33 5c 24 2e 11 48 37   d1
2591 : ae 53 eb 82 b6 b3 20 6c   96
2599 : e4 19 cb 01 8a 8b ed 9f   19
25a1 : 68 c5 c0 74 4f 7d b3 ed   36
25a9 : 19 ae 03 7b 15 3e b8 2b   c6
25b1 : 6b 32 06 ce 41 9c b0 18   7d
25b9 : a8 be f1 7b 46 2e 03 a0   cf
25c1 : 7b ef 17 b5 23 5c 13 eb   ea
25c9 : 82 b6 b3 20 6c e4 18 b8   57
25d1 : 0d 03 df 5c 48 33 5c 07   81
25d9 : b5 8a 9f 5c 15 b5 98 66   75
25e1 : b8 48 5c 22 90 7c e7 29   f8
25e9 : f5 c1 5b 59 86 2e 03 80   ba
25f1 : 45 f5 c4 81 e5 a7 c8 05   5b
25f9 : 61 63 18 b8 0d d1 ea 18   64
2601 : b8 0f 50 fa 9c 77 c3 91   6c
2609 : c7 ae ae 4f bc 03 17 01   ff
2611 : fc 3a ca 18 b8 0e d6 7d   32
2619 : 4d ca 86 2e 03 0d 65 0c   79
```

```
2621 : 5c 07 b8 9f 53 73 53 8e   5e
2629 : a1 8b 80 ff af 59 43 17   b1
2631 : 01 f8 d3 ea 6e 6a 71 df   c0
2639 : 79 72 15 d8 62 06 ce 41   60
2641 : 8b 80 f7 67 be fa d2 0c   1e
2649 : d7 01 ed 62 a7 d7 05 6d   91
2651 : 66 40 d9 c8 31 70 1f e9   4e
2659 : 3d f7 06 42 bb 0c 19 ae   3a
2661 : 12 17 08 a9 f5 c1 5b 59   c4
2669 : 90 36 72 0c 5c 07 a7 ac   28
2671 : a1 8b 80 ed a7 d4 dc be   c8
2679 : d9 c8 33 5c 24 2e 11 48   97
2681 : 3d 37 29 f5 c1 5b 59 86   cc
2689 : 7a 40 31 39 33 5c 27 be   c7
2691 : f9 24 40 fb dc 73 3d 20   ca
2699 : 18 9c 99 ae 14 77 79 43   a5
26a1 : 84 32 27 e6 81 9e 90 48   c5
26a9 : ea 01 ef b3 e4 40 ff 83   dd
26b1 : 0c f3 40 62 8a 9f 9a 0a   37
26b9 : ec 31 31 70 69 cd b9 a8   d5
26c1 : 26 77 99 c2 cb 22 29 f0   b6
26c9 : 42 62 e0 d3 b9 72 fb 94   37
26d1 : 63 1e 68 29 31 70 69 dc   79
26d9 : b9 e0 63 57 ab b0 c4 c5   a5
26e1 : c1 a7 36 e6 a0 99 de a7   82
26e9 : 0b 2c 88 99 c1 08 a9 f0   44
26f1 : 42 62 e0 d3 b9 72 fa 79   25
26f9 : 8c 79 a0 a3 1e 68 41 af   68
2701 : 98 b8 34 ee 5c f0 12 c6   03
2709 : af 63 cd 05 18 f3 42 0d   c2
2711 : 7b 1c 10 83 bd 4b 1a bc   27
2719 : ce 08 0d e9 ef bb 66 31   45
2721 : e6 82 8c 79 a1 06 bd 01   de
2729 : 1d c5 c6 af 33 5c 07 b5   6e
2731 : 8a 9f 5c 15 b5 98 67 50   a3
2739 : 06 27 26 6b 84 f7 df f0   33
2741 : 64 0c f9 10 90 b1 d0 d4   af
2749 : bd 8b 14 bc ee 0e 4c 28   49
2751 : 4d 04 a2 02 c1 c6 51 a3   68
2759 : d5 10 17 91 8c a3 49 4a   ce
2761 : 34 14 15 1a 3d 82 a2 58   4b
2769 : ca 34 cb 51 01 7f 2c 65   f2
2771 : 1a 6d 54 69 97 05 46 82   44
2779 : 82 a2 58 ca 34 [illegible] 28 d3   86
2781 : 2e 0a 8d 1e c1 51 2c 65   fe
2789 : 1a 78 2a 20 2d a6 32 8d   5a
2791 : 2c 54 40 4e 31 94 69 7c   18
2799 : a2 02 ff d6 32 8d 2e 14   87
27a1 : 40 5e 3e 32 8d 2f 14 40   09
27a9 : 5f f3 8c a3 4a 6a 20 2f   70
27b1 : cd 8c a3 48 6a 20 25 f1   d6
27b9 : 94 69 88 a3 4b 18 2a 3f   35
27c1 : 51 a5 62 88 0b 77 8c a0   6e
27c9 : c4 9d b8 0f 9d be f0 ed   db
27d1 : c1 8b 70 51 a5 e4 ec 8e   f0
27d9 : db 4f 05 46 96 33 b2 1a   68
27e1 : 58 a8 d2 fa 76 43 4b e5   1c
27e9 : 1a 73 a7 64 34 b8 51 a7   d1
27f1 : 83 3b 21 a5 e2 8d 25 4e   db
27f9 : c8 69 4d 46 98 93 b2 1a   b7
2801 : 43 51 a5 7c ec 86 98 8a   60
2809 : 34 da ce c8 69 58 a3 48   ef
2811 : 73 b2 0c d9 80 c4 e5 3e   5e
2819 : b8 2b 6b 30 64 4d 50 19   6c
2821 : 13 c9 0e 38 ac 53 f1 01   d3
2829 : a1 37 10 50 23 84 46 15   cf
2831 : 3e fc 1a 4a f7 66 78 80   53
2839 : d2 ba e7 81 92 67 88 19   4b
2841 : dd ff 0b 2c 88 c7 e0 64   79
2849 : d6 31 92 0f bf f6 98 c8   e6
2851 : 9e 48 40 1c 75 76 18 34   7b
2859 : fd a2 52 db 34 25 d6 ec   59
2861 : 71 e5 b6 68 f6 e9 43 17   79
2869 : 09 1e 18 1f 53 72 a1 bc   34
2871 : c4 04 46 f5 37 2f ab 93   4a
2879 : 55 c5 28 0a 59 4d f9 e5   b0
2881 : 18 b8 6b 29 37 e8 94 62   c7
2889 : e1 3e a6 e5 43 79 88 08   22
2891 : 8d ea 6e 5f 0e 3a ae 29   db
2899 : 40 52 c8 cd 70 91 e9 06   35
28a1 : a3 e9 91 e0 7b 96 46 2e   9b
28a9 : 03 ee 6b 3e 9d 1e 06 de   67
28b1 : 39 8b 86 77 89 c2 cb 22   62
28b9 : 24 18 a2 c7 fc 75 fc 98   2b
28c1 : fe cc 7f ea 8f f2 c7 87   21
28c9 : ee 47 f7 1a fb 96 be e7   db
28d1 : ef f7 3d 5f 73 67 ee 7f   24
28d9 : 4f b9 eb fb 97 db ee d6   41
28e1 : fa f7 7c 62 e1 9d e2 70   ba
28e9 : b2 c8 89 07 58 e6 3f e3   c4
```

```
28f1 : af f9 f7 dd fe fb 87 be   c2
28f9 : c1 df 7f 7e fb 89 be da   16
2901 : bc f6 1e 3f 79 d2 79 f4   a6
2909 : bc d1 79 e3 bc ea 3c cf   3c
2911 : 78 2b 23 17 01 8a 2c 7b   d7
2919 : fb 1f cf 9b d5 cd ea e6   50
2921 : f5 73 7a b9 bd 5c de ae   3d
2929 : 6f 57 36 02 96 46 2e 03   6c
2931 : f0 5c c7 bf b1 ea e6 f5   33
2939 : 73 7a b9 bd 5c de ae 6f   66
2941 : 57 37 ab 9b d5 cd 80 a5   ab
2949 : 91 8b 80 eb 1c c7 bf b1   a0
2951 : e9 79 ff 7f cf 87 9d 47   25
2959 : 3b cb ce ef f3 8a ce 7d   f5
2961 : 9d 01 4b 23 17 01 e5 39   3a
2969 : 8f 7f 63 f9 f3 af 33 2f   b8
2971 : 33 2c 33 2c 33 25 e6 79   f8
2979 : 19 9f 1e 64 05 2c 8c 5c   13
2981 : 07 90 e6 3d fd 8d e6 15   44
2989 : e6 15 e6 15 e6 15 e6 15   33
2991 : e6 15 e6 15 e6 14 05 2c   da
2999 : 8c 5c 06 63 98 f7 f6 37   d5
29a1 : 98 57 98 57 98 57 98 57   4b
29a9 : 98 57 98 57 98 57 98 50   45
29b1 : 14 b2 31 70 1c c7 31 ef   1d
29b9 : ec 6f 30 af 30 af 30 af   ff
29c1 : 30 b3 70 b3 70 b8 18 5c   43
29c9 : 8c 28 0a 59 18 b8 0d 07   a1
29d1 : 31 ef ec 6d b0 8c f2 bc   97
29d9 : bf 2b e2 f2 be 9f 2a 5f   95
29e1 : 95 f2 79 5f 63 ca 80 a5   13
29e9 : 91 8b 80 e9 39 8f 7f 63   72
29f1 : e4 f9 5e 4f 94 a7 84 cf   8c
29f9 : 85 fe fe 15 bf 85 c8 f0   8d
2a01 : bf 9b c2 80 a5 91 8b 80   65
2a09 : db 39 8f 7f 62 9a 0c 20   c0
2a11 : b2 0b 21 de a1 e2 21 b3   8a
2a19 : 43 fe 90 80 a5 91 8b 80   26
2a21 : ed 1c c7 bf b1 fe f4 3c   65
2a29 : 54 3c bb 0f 6a c3 7d 12   4b
2a31 : 0c 4c 58 89 44 80 a5 91   ad
2a39 : 8b 80 d2 73 1e c6 d6 ca   ca
2a41 : 25 ac 4b 58 96 b1 2d 62   0b
2a49 : 69 44 d2 89 a5 12 02 96   da
2a51 : 46 2e 03 fa 1c c7 bf b1   31
2a59 : d2 88 f1 66 cb 36 59 b2   f2
2a61 : cd 96 6c b3 65 a0 29 64   d4
2a69 : 62 e0 3a b7 31 ef ec 5b   be
2a71 : 2c d9 66 cb 36 59 b2 cd   31
2a79 : 96 6c b3 65 a0 29 64 62   88
2a81 : e0 3f 45 cc 7b fb 16 cb   73
2a89 : 36 59 b2 cd 96 6c b3 65   38
2a91 : 9b 2c d9 68 0a 59 18 b8   03
2a99 : 0f b6 e6 3d fd 8b 65 9b   6e
2aa1 : 2c d9 66 cb 36 59 b2 cd   61
2aa9 : 96 6c b4 05 2c 8c 5c 07   ea
2ab1 : dc 73 1e fe c6 82 d4 16   ae
2ab9 : a0 b5 05 a8 2d 41 6a 0b   27
2ac1 : 50 5a 02 96 46 2e 03 fe   72
2ac9 : 0e 63 df d8 d0 5a 82 d4   2f
2ad1 : 16 a0 b5 05 a8 2d 41 6a   13
2ad9 : 0b 40 52 c8 c5 c0 7f 8d   2d
2ae1 : cc 7b fb 1a 0b 50 5a 82   4f
2ae9 : d4 16 a0 b5 05 a8 2d 41   74
2af1 : 68 0a 59 18 b8 0f cd 73   da
2af9 : 1e fe c6 82 d4 16 a0 b5   84
2b01 : 05 a8 2d 41 6a 0b 50 5a   c3
2b09 : 02 96 46 2e 03 07 31 ef   bb
2b11 : ec 6a d6 ef d6 ee 2c 52   20
2b19 : cf 96 fd 6b 7f aa d0 e7   80
2b21 : 40 52 c8 c5 c0 7b 87 31   dd
2b29 : ef ec 7d 89 c8 ce e6 4e   5a
2b31 : e0 4e 73 3b b7 3b f7 ce   4f
2b39 : c9 9d 01 4b 23 17 01 fe   68
2b41 : 67 31 ef ec 7a d3 ba d3   b3
2b49 : ba d3 ba d3 ba d3 ba d3   f3
2b51 : ba d3 ba d3 a0 29 64 62   c7
2b59 : e0 3f da e6 3d fd 8f 5a   23
2b61 : 77 5a 77 5a 77 5a 77 5a   0b
2b69 : 77 5a 77 5a 77 5a 74 05   5d
2b71 : 2c 8c 5c 07 f5 1c c7 bf   ba
2b79 : b1 de 4e de 4e de 4e de   dc
2b81 : 4e de 4e de 4e de 4e de   81
2b89 : 4e 80 a5 91 8b 80 ff 9d   ab
2b91 : cc 7b fb 1d e4 ed e4 ed   ea
2b99 : e4 ed e4 ed e4 ed e4 ed   98
2ba1 : e4 ed e4 e8 0a 59 18 b8   0f
2ba9 : 0f ff 6e 63 df d8 ef 27   93
2bb1 : 6f 27 6f 27 6f 27 6f 27   b1
2bb9 : 6f 27 6f 27 6f 27 40 52   52
```

```
2bc1 : c8 c5 c0 7f b9 cc 7b fb  74
2bc9 : 1d e4 ed e4 ed e4 ed e4  f8
2bd1 : ed e4 ed e4 ed e4 ed e4  d0
2bd9 : e8 0a 59 18 b8 0f fa dc  c9
2be1 : c7 bf b1 fe e9 df aa 77  0b
2be9 : ea 9d fa a7 7f a4 ef 32  97
2bf1 : 0f da 83 e3 c1 80 a5 91  a4
2bf9 : 8b 80 ff 4b 98 f7 f6 3a  c7
2c01 : f0 79 70 79 70 79 70 79  81
2c09 : 70 79 70 79 70 79 70 60  d6
2c11 : 29 64 62 e0 3f c2 e6 3d  41
2c19 : fd 8d 63 fa c7 f5 8f eb  57
2c21 : 1f d6 3f ac 7f 58 fe b1  2b
2c29 : fc 05 2c 8c 5c 07 fa dc  e8
2c31 : c7 bf b1 ac 7f 58 fe b1  f4
2c39 : fd 63 fa c7 f5 8f eb 1f  69
2c41 : d6 3f 63 90 0f b8 f6 4a  c9
2c49 : c2 c6 35 2d a7 32 07 fd  85
2c51 : 26 39 49 f7 e0 66 cc 24  22
2c59 : 75 41 c9 06 f6 7b eb 3b  13
2c61 : 28 1f d1 64 67 54 06 27  99
2c69 : 29 f5 c1 5b 59 88 11 c5  12
2c71 : ab 42 6e 2d 10 16 49 b6  c3
2c79 : a9 38 45 5d 86 0c 5c 07  84
2c81 : 5c 7b ee dd 91 a8 9a 8f  fa
2c89 : 1b 65 43 68 8a b4 9f c0  83
2c91 : fe 8b 24 08 e2 d4 62 4e  5a
2c99 : dc 24 2e 1b ef 0e dc 15  83
2ca1 : d8 60 db 53 4f 49 c5 af  c0
2ca9 : 6c db 54 04 f1 27 16 a6  96
2cb1 : a2 6d aa 5c 6d e9 cd 5d  58
2cb9 : 29 2e d1 a1 2e aa 71 df  60
2cc1 : 6e 6c 8d 44 db 54 9c 22  68
2cc9 : d0 5b 8b 48 1f 0d 91 a8  25
2cd1 : 9a 8f 1b 65 43 68 8a b4  b1
2cd9 : 9f c0 46 c9 a4 af 75 01  43
2ce1 : 13 25 01 13 26 b2 47 ca  d4
2ce9 : 1d 97 0e cd 76 6b be 36  39
2cf1 : cb a5 b4 76 67 ca 03 21  a6
2cf9 : ef bd bb 24 1e cf b4 70  4e
2d01 : ee 12 1b 66 b3 f1 cb b4  ef
2d09 : c5 2e 8b 5b 9b 4d 6d 47  9c
2d11 : 6c bb 9e d9 8c 98 48 53  93
2d19 : 2d ad 04 5a cf 45 dd a4  51
2d21 : f8 ac 38 44 cf 94 06 23  06
2d29 : df 78 4c 21 78 5b 5a 08  57
2d31 : b5 9e 8b bb 49 ec 38 46  f9
2d39 : 11 ae cd 45 2c 98 50 a4  cf
2d41 : 7b 61 d9 70 ec d7 66 bb  90
2d49 : e3 6c ba 5b 47 66 7b 60  d3
2d51 : 32 1e fb e7 61 07 b3 ed  87
2d59 : 1c 3b 42 c4 b6 b4 68 da  a4
2d61 : 51 8a c3 84 4c f6 c0 62  bd
2d69 : 3d f7 ef 61 0f 04 b6 b4  1f
2d71 : 68 da 51 61 c2 26 bb 35  7e
2d79 : 14 9c 3b 35 de 4c 28 48  d2
2d81 : 68 6f ef 62 bb b9 b2 d4  e7
2d89 : a3 68 da 8d ee b6 b2 c5  c4
2d91 : ef ec 8d 65 27 0e cd 77  0f
2d99 : 93 0a 14 0c 51 b4 ac fd  21
2da1 : fe 22 7e 88 73 5c 6f fc  33
2da9 : 44 fd 00 f1 13 e3 05 61  51
2db1 : 9e 88 0c 87 be fe 76 23  8b
2db9 : 42 42 c7 43 15 9b 4d 4d  75
2dc1 : 4e 92 d9 32 3d 10 fa e8  27
2dc9 : 3d e4 bc c9 37 7f 92 80  9b
2dd1 : 9e 64 c8 f4 03 eb a0 f7  74
2dd9 : 92 f3 26 47 18 3e ba 19  68
2de1 : 7c 92 f8 3d 3c 98 50 31  b8
2de9 : 46 66 b8 0c 51 53 d7 06  2d
2df1 : 8f 7b ff 51 c1 f5 86 9e  8b
2df9 : 81 1b be 9b 31 01 11 c1  0e
2e01 : e6 9f d8 a8 4d 70 8c ae  ea
2e09 : c3 12 17 04 fc 50 57 61  8e
2e11 : 83 17 0e 39 04 89 24 78  d9
2e19 : a1 14 92 3c 90 72 9f 8a  21
2e21 : 11 95 d8 60 c5 c3 8e 41  76
2e29 : 22 9b 70 57 61 83 17 09  c0
2e31 : 0d c2 29 24 79 20 e5 36  0b
2e39 : e1 18 ce b8 48 d7 03 df  00
2e41 : 6d e6 46 33 5c 06 28 a6  fd
2e49 : 75 c0 f7 df 66 64 0d 86  e3
2e51 : 0c eb 84 8d 70 d6 7d e3  a1
2e59 : cc 33 5c 06 28 a9 eb 81  19
2e61 : 9d 70 18 9c ef bc 3d 70  8b
2e69 : 68 f7 bf f5 1c 1f 59 db  53
2e71 : 33 5c 06 27 29 eb 84 0d  57
2e79 : 86 1a 3d f4 5a 3d 51 a5  1a
2e81 : 8a 8d 1e a8 6b 28 65 bc  75
2e89 : d4 79 45 2e b2 bb 46 15  7d
2e91 : 2e 61 9e 28 0c 4e 53 d7  4d
2e99 : 03 3a e1 23 5c 35 1f 7f  81
2ea1 : 5c c8 16 b3 20 65 0d 5d  79
2ea9 : 86 24 2e 09 fc 00 ae c3  00
2eb1 : 13 17 06 9c db 9a 82 67  d0
2eb9 : 79 9c 2c b2 22 9f 04 26  5d
2ec1 : 2e 0d 3b 97 2f b4 33 66  6a
2ec9 : 2e 0d 3b 97 3c 07 1f 12  dc
2ed1 : bb 0c 4c 5c 1a 73 6e 6a  fd
2ed9 : 09 9d ea 70 b2 c8 89 9c  4a
2ee1 : 10 8a 9f 04 26 2e 0d 3b  1d
2ee9 : 97 2f ac f3 66 2e 0d 3b  44
2ef1 : 97 3c 03 7e 2e 2b 8f 89  c5
2ef9 : 8e 08 41 de 9b f1 71 5c  7f
2f01 : 7c 4a de cc 31 b8 0f 6a  dd
2f09 : e4 92 3c 90 8a 4c ee 6f  fd
2f11 : 0b 2c 88 c7 51 8f 80 27  2f
2f19 : 6f cc 6e 03 da b9 24 8f  15
2f21 : 24 22 93 3b 9b c2 cb 22  e6
2f29 : 31 d4 37 e2 7e fe 3d c4  19
2f31 : 9d b8 71 c8 2d f7 87 6e  2d
2f39 : 0a ec 30 69 e8 09 e5 ae  be
2f41 : ef f9 66 9a 7a 02 5a d5  e7
2f49 : 43 3d 20 18 9c 99 ae 13  ad
2f51 : ea 6e 5f 7f 2c de dc 0c  7f
2f59 : a1 c6 18 93 e4 04 66 17  0c
2f61 : 05 b2 2a 13 e2 83 07 82  18
2f69 : d1 d4 99 c5 06 98 d7 35  b2
2f71 : 2e df 7a f9 ac 2e 10 54  92
2f79 : 60 f0 83 25 27 15 66 1e  c8
2f81 : 03 13 94 cf 08 17 9e 7b  d8
2f89 : 0b 84 15 10 12 69 30 78  3c
2f91 : 41 92 92 0e 8e 93 8a b3  17
2f99 : 0f 01 89 ca 67 81 8b 80  96
2fa1 : e1 eb 3e ca f3 e0 77 be  02
2fa9 : 7b 07 82 d1 f2 a4 71 42  21
2fb1 : d1 f2 de 7b 07 84 19 26  68
2fb9 : d5 a4 e2 ac c3 c0 62 72  df
2fc1 : 99 e1 03 bd f3 d8 3c 20  fb
2fc9 : c9 40 49 f1 50 14 74 8d  d5
2fd1 : ab 49 c5 59 87 80 c4 e5  19
2fd9 : 33 c1 5c 87 f1 8c d7 01  da
2fe1 : ca 8a 9f 5c 15 b5 98 ad  20
2fe9 : ec c3 1b 84 8f 24 22 90  d2
2ff1 : 7b 57 24 ce e6 [illegible] b2 c8  4d
2ff9 : 8a 6d c0 d4 50 16 1a 4c  bb
3001 : 37 08 2f d2 40 49 a4 e1  07
3009 : 16 1b 82 db fa 84 d9 03  0a
3011 : 19 01 89 c5 ca 76 e0 32 6f  7d
3019 : b8 f5 d5 df 5a dc 0c 3c  72
3021 : 24 5b 83 98 f0 64 9a 8f  a2
3029 : 15 01 3c 49 c2 3d b3 0f  fa
3031 : 09 0c 83 94 cf 03 51 35  78
3039 : 1e 36 ca 82 02 8e 93 84  61
3041 : 55 a4 fe 31 a8 a0 2f c4  a4
3049 : 93 07 84 19 29 20 2f 21  37
3051 : 26 0f 08 32 52 40 5b 24  24
3059 : 8c 3c 06 27 31 e0 c9 49  3e
3061 : c2 26 1e 03 7a e5 33 c1  45
3069 : 5a 4f e3 1a 8a 02 fc 49  e6
3071 : 30 78 41 92 92 02 f2 12  a9
3079 : 60 f0 83 25 24 05 b2 48  9d
3081 : c3 c0 6f 5c c7 83 25 24  81
3089 : 04 6a 4e 11 30 f0 18 9c  9c
3091 : c7 83 24 d4 78 ae 11 30  5f
3099 : f0 1c a7 29 9e 0a d2 7f  2b
30a1 : 18 d4 50 17 d7 49 c2 2a  41
30a9 : d2 7f 18 d4 50 17 d7 49  8b
30b1 : 83 c2 0c 94 9c 22 61 e0  4d
30b9 : 31 39 4c f0 56 93 f8 c6  2b
30c1 : a2 80 b9 09 38 45 5a 4f  e9
30c9 : e3 0c 49 f2 02 31 a8 a0  f1
30d1 : 2f 35 26 0f 08 32 52 40  e2
30d9 : 4f 12 30 f0 18 9c c7 83  e8
30e1 : 25 27 08 98 78 0d eb 94  78
30e9 : cf 05 69 3f 8c 6a 28 0b  50
30f1 : 78 93 07 84 19 29 38 44  ca
30f9 : c3 c0 62 72 99 e0 ae c3  e6
3101 : 06 9e 80 96 b5 50 d3 d0  18
3109 : 13 cb 5a 9b 9a 84 f2 40  26
3111 : d3 d0 12 cd e6 64 86 3b  ad
3119 : ef 97 3a 07 83 98 6a 26  36
3121 : 9e 93 84 55 d8 60 d3 d0  d6
3129 : 12 d6 af bc 1c c8 1f 2e  0b
3131 : 72 b4 9f c6 31 70 1a 87  cc
3139 : ef 5a 65 d9 97 1a f7 80  15
3141 : 6a 3c 57 08 ab 7b 31 5f  ba
3149 : cf 8d 37 1a 51 96 e0 8f  5c
3151 : 65 a3 de 61 a8 f1 5c 22  3b
3159 : ad 27 f1 8d 45 01 68 a4  0f
3161 : c1 e1 06 4a 46 2e 03 a9  13
3169 : 3f 7a d3 2e cc b8 d7 bc  0b
3171 : 04 9c 22 61 e0 31 39 4c  8d
3179 : f0 57 61 83 17 01 bd 72  33
3181 : 4c ee 6f 0b 2c 88 a6 c8  b5
3189 : 0c 49 b6 09 0c 86 fb c3  75
3191 : 6c 0a de cc 57 f3 e3 4d  93
3199 : c6 94 65 b8 23 d9 68 f7  ac
31a1 : 98 6a 3c 54 04 f1 27 08  84
31a9 : f6 cd 44 d4 78 db 2a 1b  77
31b1 : 44 55 a4 fe 31 a8 a0 2e  e0
31b9 : b2 4e 11 56 93 f8 c6 a2  03
31c1 : 80 be d2 4e 11 56 93 f8  23
31c9 : c6 a2 80 9f 24 c1 e1 06  d8
31d1 : 4a 48 0b 70 93 84 4c 3c  17
31d9 : 06 27 29 9e 0a ec 30 67  29
31e1 : 2c 06 2a 2f b8 42 cc 31  b4
31e9 : 70 1d e1 ef b8 42 cc 1d  69
31f1 : c2 6b 81 9a e1 21 70 8a  1a
31f9 : 41 bd 72 9e fc 38 f5 d5  9e
3201 : c9 aa 0a d2 7f 18 c5 c0  4e
3209 : 7f 92 2c 7e 02 02 4d 26  5e
3211 : 0f 08 32 52 40 51 d2 71  b8
3219 : 56 61 e0 31 39 4c f0 57  e6
3221 : 21 fc 63 17 01 fe 48 b1  09
3229 : f8 08 09 34 8c 3c 06 27  ff
3231 : 31 e0 c9 49 01 3c 49 83  8c
3239 : c2 0c 94 90 14 74 9c 55  3a
3241 : 98 78 0d eb 94 cf 05 72  97
3249 : 1f c6 30 f0 1b d7 31 e0  ec
3251 : c9 41 ee aa 7c 54 04 f1  2a
3259 : 23 0f 01 89 cc 78 32 52  73
3261 : 40 4f 12 60 f0 83 25 24  e1
3269 : 05 1d 27 15 66 1e 03 94  f6
3271 : e5 33 c1 5c 87 f1 95 2e  a6
3279 : 61 87 80 c5 15 33 c1 83  70
3281 : c1 43 3c 50 1e d6 29 24  82
3289 : 79 21 14 99 de 26 83 b9  6c
3291 : 3c 14 63 c1 af 7f 25 52  19
3299 : e6 18 78 0c 51 53 3c 18  fc
32a1 : 3c 14 33 c5 01 ed 62 92  9b
32a9 : 47 92 11 49 9d e2 68 3b  b0
32b1 : 93 c1 46 3c 1a f7 f2 55  16
32b9 : d8 62 32 b7 b3 0c 6e 03  a1
32c1 : da c5 24 8f 24 22 93 3b  91
32c9 : c4 d0 77 27 82 8a 6f c1  76
32d1 : 87 e1 06 bd 83 c2 0c 94  2a
32d9 : 90 14 74 9c 55 98 78 0c  38
32e1 : 4e 53 3c 15 c8 7f 18 d4  1d
32e9 : 50 17 f6 a4 61 d0 37 ae  6e
32f1 : 63 c1 92 91 87 80 c4 e6  69
32f9 : 3c 19 29 20 28 e9 20 2f  c1
3301 : 21 26 0f 08 32 52 70 89  85
3309 : 87 80 e5 39 4c f0 56 93  3e
3311 : f8 c3 12 7c 80 8c 31 27  7e
3319 : c8 03 51 40 4b 24 e1 15  47
3321 : 69 3f 8c 67 78 03 d4 3d  a7
3329 : f7 f0 f2 b8 f5 d5 c9 d8  53
3331 : 04 03 bc a3 51 40 5e aa  40
3339 : 4c 1e 10 64 a4 80 a7 24  5a
3341 : e1 13 0f 01 89 ca 67 82  21
3349 : b4 9f c6 35 14 05 ea a4  83
3351 : c1 e1 06 4a 48 09 e2 4e  c3
3359 : 11 30 f0 18 9c a6 78 2b  f9
3361 : 49 fc 63 51 40 5e aa 4c  66
3369 : 1e 10 64 a4 e1 13 0f 01  32
3371 : 89 ca 67 82 b4 9f c6 35  57
3379 : 14 05 ea a4 e1 15 69 3f  4a
3381 : 8c 6a 28 0b d5 48 c5 c0  e6
3389 : 77 a7 97 2a cc b9 74 b8  dd
3391 : 09 30 78 41 92 92 0f 63  b9
3399 : ce 49 c2 26 1e 03 13 94  f1
33a1 : cf 05 69 3f 8c 6a 28 0b  08
33a9 : 92 93 07 84 19 29 20 27  01
33b1 : 89 20 21 a4 80 86 92 02  b2
33b9 : 9c 93 84 58 3c 20 c9 da  ed
33c1 : 5a 98 78 0c 4e 53 3c 15  a2
33c9 : a4 fe 31 a8 a0 2f 55 26  73
33d1 : 0f 08 32 52 40 4b 24 80  ab
33d9 : bc 04 8d b5 49 c2 26 1e  31
33e1 : 03 13 94 cf 05 69 3f 8c  3f
33e9 : 6a 28 0b c0 49 83 c2 0c  16
33f1 : 94 9c 22 61 e0 31 39 4c  9d
33f9 : f0 56 93 f8 c6 a2 80 bc  15
3401 : 04 8d b5 49 c2 2a d2 7f  2a
3409 : 18 cc c0 18 a2 a7 98 06  90
3411 : 72 c0 62 a2 fb fb bc 23  a9
3419 : 33 00 62 72 9e 60 18 b8  f2
3421 : 0c 51 53 5c 20 7f 37 85  1c
3429 : 19 02 38 ab 90 78 26 78  1d
```

Listing. »Basic-Maker« (Fortsetzung)

```
3431 : 18 78 48 cc 07 3b ee 3d  b1
3439 : 75 77 d6 3c 18 3c 20 c9  1e
3441 : 40 4f 38 a6 d5 a4 e2 af  79
3449 : b6 67 78 03 74 7b ea 68  d1
3451 : 30 78 41 92 83 d7 9c 7e  c6
3459 : 29 b5 69 38 ab 30 f0 18  ee
3461 : 9c a6 78 1a 8a 02 ef 12  4e
3469 : 36 ad a3 de 62 4e 11 19  4a
3471 : 13 ef 00 64 4f 30 15 a4  1d
3479 : fe 30 dd 27 de 04 63 51  2a
3481 : 40 57 e9 38 45 5a 4f e3  1a
3489 : 1a 8a 02 fe 09 30 78 41  bf
3491 : 92 93 84 4c 3c 06 27 29  7a
3499 : 9e 0a d2 7f 18 d4 50 17  78
34a1 : 05 26 0f 08 32 52 40 4f  d4
34a9 : 12 30 f0 18 9c c7 83 25  73
34b1 : 27 08 98 78 0d eb 94 cf  34
34b9 : 05 69 3f 8c 6a 28 0b 82  ed
34c1 : 93 07 84 19 29 20 27 89  5f
34c9 : 18 78 0c 4e 63 c1 92 92  9e
34d1 : 02 78 91 87 80 de b9 8f  6a
34d9 : 06 4a 4e 11 30 f0 1c a7  04
34e1 : 29 9e 0a d2 7f 18 d4 50  e3
34e9 : 17 41 27 08 ab 49 fc 63  2b
34f1 : 51 40 5c d4 9c 22 ad 27  f4
34f9 : f1 87 a8 9f 78 11 90 23  64
3501 : 8a b9 07 82 67 81 87 84  24
3509 : 8c c0 73 be e3 d7 57 7d  ff
3511 : 63 c1 83 c2 0c 94 04 f3  eb
3519 : 8a 6d 5a 4e 2a fb 66 77  c5
3521 : 80 37 47 be 2e c0 d4 50  c3
3529 : 14 24 98 3c 20 c9 48 da  24
3531 : b4 9c 22 61 e0 31 39 4c  fd
3539 : f0 81 a9 60 c5 80 3e 6b  90
3541 : 60 c1 e1 06 4a 02 9c 91  05
3549 : b5 6d 1e f3 12 71 56 61  84
3551 : e0 31 39 4c f0 19 13 b0  27
3559 : 03 51 40 45 24 6d 5a 4e  71
3561 : 11 19 13 ef 00 64 4f 30  82
3569 : 15 a4 fe 31 9c b0 18 8f  85
3571 : 7d e8 58 40 da 58 1a 8a  6e
3579 : 02 7c 93 07 84 19 29 20  75
3581 : 2d c2 4e 11 30 f0 18 9c  e9
3589 : a6 78 19 ae 03 7b 15 3d  62
3591 : f8 71 eb ab 93 54 06 44  2f
3599 : fb f0 56 93 f1 f2 93 5c  d2
35a1 : 06 24 f2 42 34 0f db 60  aa
35a9 : 31 27 c8 03 51 40 4b 24  8d
35b1 : e1 15 69 3f 8c c1 e1 06  c9
35b9 : 4e d2 ac c3 c0 62 72 99  30
35c1 : e0 c5 80 3e f5 ec 18 3c  0b
35c9 : 20 c9 40 53 92 36 ad 27  28
35d1 : 15 66 1e 03 13 94 cf 01  18
35d9 : 91 3b 00 33 bc 01 ba 3d  a8
35e1 : f6 54 43 51 40 50 92 60  8e
35e9 : f0 83 25 24 04 f1 23 6a  9a
35f1 : d2 70 89 87 80 c4 e5 33  7b
35f9 : c2 00 e2 1a 8a 02 29 23  5b
3601 : 6a d2 70 88 c8 9f 78 0a  81
3609 : d2 7f 18 62 4f 90 06 a2  c4
3611 : 80 96 49 c2 2a d2 7f 18  ee
3619 : cc c0 18 a2 a7 98 11 86  30
3621 : 24 f9 00 6a 28 09 64 9c  25
3629 : 22 ad 27 f1 8c cc 01 8a  72
3631 : 2a 79 81 18 ce 58 0c 47  ea
3639 : be 73 12 07 ed 88 62 e0  85
3641 : 3a 07 be bf 88 6a 28 09  35
3649 : f2 4c 1e 10 64 a4 80 b7  c8
3651 : 09 20 25 92 40 5d 64 9c  c0
3659 : 22 62 e0 3a 07 be fd b1  ed
3661 : 0c 3c 06 27 29 9e 12 35  2c
3669 : c1 3d f8 71 eb ab 93 54  48
3671 : 06 44 fb f0 56 93 f8 c3  24
3679 : 12 7c 80 35 14 04 b2 4e  59
3681 : 11 56 93 f8 c6 2e 03 f0  8d
3689 : a2 c7 e0 38 ab 31 70 1b  8a
3691 : 73 df 7d 88 88 08 1c 55  48
3699 : 9a 89 b5 69 38 45 5d 86  c3
36a1 : 0c 5c 06 43 df 7b 71 20  a5
36a9 : 26 b0 c4 9d b8 48 5c 37  ba
36b1 : de 1d b8 30 78 41 92 6a  03
36b9 : 3c 57 08 98 78 0c 4e 53  7e
36c1 : 3c 0c b7 09 0b 84 f7 c6  55
36c9 : 2c 6a 28 09 e2 4e 11 ed  16
36d1 : ab 49 fc 63 51 40 57 69  14
36d9 : 30 78 41 92 93 84 4c 3c  ef
36e1 : 06 27 29 9e 0a d2 7f 18  fe
36e9 : 62 4f 90 06 a2 80 96 49  f3
36f1 : c2 2a d2 7f 18 c5 c0 7d  1b
36f9 : c7 31 e0 d7 ed 2a d5 c8  a5
3701 : 7f 19 5d 86 0c 5c 06 28  41
3709 : b1 e0 d7 ed 2a d5 c8 7f  51
```

```
3711 : 18 62 4f 90 11 86 24 f9  0a
3719 : 01 18 62 4f 90 11 95 d8  42
3721 : 60 ce 58 0c 47 be fc ab  36
3729 : 40 dd ac 62 e1 9d cd e1  d5
3731 : 65 91 1c 44 d7 03 17 01  e2
3739 : bd 73 1d 6d fd 42 7f 00  95
3741 : 20 28 ee 2a c6 24 ed c2  41
3749 : 47 c0 1b ef 0e dc 18 3c  56
3751 : 20 c9 40 4f 38 a6 d5 a4  a9
3759 : e2 ac c3 c0 62 72 99 e1  7e
3761 : db 31 70 1b d7 31 e0 bf  5e
3769 : 40 49 a4 6d 5b 47 bc c4  91
3771 : 9c 55 ab 90 fe 30 c4 9f  78
3779 : 20 23 2b b0 c4 85 c3 b2  f9
3781 : d1 ef 33 69 56 ae 43 f8  1e
3789 : ca de cc 31 b8 48 f2 42  3a
3791 : 2f 65 a3 de 66 d2 ad 5c  a4
3799 : 87 f1 b3 af ec c8 42 27  68
37a1 : 6e 19 d7 f6 79 64 26 fb  bc
37a9 : c3 b7 05 76 18 62 dc 14  88
37b1 : 61 70 d1 ed c8 27 d5 ec  73
37b9 : 98 4f aa 32 4f 2d ed 2e  5c
37c1 : 66 d2 ad 5c 87 f1 86 24  f2
37c9 : f9 01 19 01 57 38 ab 57  3e
37d1 : 21 fc 64 1e c5 45 c5 5a  a0
37d9 : b9 0f e3 20 f5 17 6e 2a  3d
37e1 : d5 c8 7f 18 62 4f 90 11  02
37e9 : 86 24 f9 01 19 09 48 bb  92
37f1 : 92 e2 ad 5c 87 f1 8c 5c  de
37f9 : 06 bb 9e cb 47 bc cd a5  db
3801 : 5a b9 0f e3 18 b8 0f ea  d1
3809 : 39 8f 06 bc d4 78 a8 0b  ed
3811 : c8 49 83 c2 0c 94 9c 22  d3
3819 : 61 e0 31 39 4c f0 56 93  2b
3821 : f8 ca ec 30 62 e0 31 45  3c
3829 : 8f 06 bc d4 78 a8 0b c8  0f
3831 : 49 83 c2 0c 94 9c 22 61  e7
3839 : e0 31 39 4c f0 56 93 f8  8c
3841 : c3 12 7c 80 35 14 04 b2  a6
3849 : 4e 11 56 93 f8 ca ec 30  22
3851 : 67 2c 06 23 df 5e 41 81  ad
3859 : ca 82 62 e1 9d cd e1 65  d4
3861 : 91 1c 44 d7 03 17 01 bd  75
3869 : 73 1d 6d fd 42 [illegible] 00 30  06
3871 : 78 41 92 83 d8 fe 47 15  6c
3879 : c5 59 87 80 c4 e5 33 c0  a6
3881 : 62 4e dc 24 7c 01 be f0  73
3889 : ed c1 83 c2 0c 94 04 f3  ed
3891 : 8a 6d 5a 4e 2a cc 3c 06  38
3899 : 27 29 9e 1d b3 17 01 bd  14
38a1 : 73 1e 0b f3 51 e2 a0 24  5c
38a9 : d2 36 ad a3 de 62 4e 11  d2
38b1 : 56 93 f8 c3 12 7c 80 35  f9
38b9 : 14 04 b2 4e 11 56 93 f8  4a
38c1 : c6 a2 84 a4 5d c9 49 01  d9
38c9 : 79 09 30 78 41 92 93 84  e2
38d1 : 4c 3c 06 27 29 9e 0a d2  f7
38d9 : 7f 19 01 0d c5 5a bb 0c  fd
38e1 : 0c 49 db 84 85 c3 7d e1  49
38e9 : db 82 bb 0c 1b 56 69 e9  54
38f1 : 38 ab ed 9b 56 80 86 93  98
38f9 : 8a b5 72 1f c6 41 e8 c6  86
3901 : e2 ad 5c 87 f1 90 10 dc  5f
3909 : 55 8c 49 db 81 8b 80 f4  d2
3911 : dc ef bc 3b 70 57 61 86  d0
3919 : 17 0e 01 b5 7c 57 15 7d  00
3921 : b3 6a d0 10 d2 71 56 ae  af
3929 : 43 f8 c6 2e 03 fb 0e 7b  9f
3931 : 2d 1e f3 36 95 6a e4 3f  f0
3939 : 8c 61 e0 31 45 4c f0 60  0f
3941 : f0 50 da b7 f2 58 3c 14  12
3949 : 63 8a 13 a3 8c 89 f1 42  8c
3951 : 33 0a 83 ee f5 f9 9a e1  a5
3959 : 23 24 22 92 46 fc 27 d4  fc
3961 : 26 a8 30 a8 3e f8 9f 9a  5c
3969 : 8a 02 59 27 08 c6 97 66  11
3971 : d0 97 4f bf dd fa bb fa  73
3979 : 50 30 df cd c6 95 ca 83  de
3981 : 29 1e cb 47 bc c4 04 5a  4c
3989 : 59 2a ef e9 42 63 bc 07  71
3991 : df 91 f8 f6 a9 f2 01 58  3d
3999 : 58 cc 66 03 ee 83 f1 f3  0c
39a1 : 93 e4 02 b0 b1 98 b0 07  ee
39a9 : cb bf 1f 82 9f 20 15 85  c6
39b1 : 8c c1 e0 fb fb 9f 8e c5  58
39b9 : 3e 40 2b 0b 18 64 4c f0  fb
39c1 : 33 bf 01 bd 3d 43 a0 ad  98
39c9 : 79 50 66 b8 4f 50 e8 3f  b5
39d1 : c5 6b ca 83 35 c3 51 4e  c2
39d9 : 54 19 52 25 84 fa 9b 9a  b7
39e1 : 9c 7a 9b 97 df ff bf 1d  cb
39e9 : c2 7c 80 33 c4 01 ed 62  41
```

```
39f1 : ef 8d 47 8d 51 65 15 3f  3d
39f9 : 10 0d 45 01 2c dd f6 86  9c
3a01 : 32 02 fc 6e 11 99 e2 03  b1
3a09 : 4a eb 9e 01 9e 20 67 77  88
3a11 : fc 2c b2 23 1f 80 94 b8  ee
3a19 : 27 63 59 ef 5a 57 5c f0  fa
3a21 : 12 97 04 ec 6b 3c ac ee  c7
3a29 : ff 85 96 44 63 f0 12 35  89
3a31 : 14 90 11 12 70 89 9d f8  d0
3a39 : 0e 55 15 0e 82 b5 e5 41  e9
3a41 : 9a e1 a9 53 72 fa af 18  8e
3a49 : e8 3f c7 71 06 06 eb 18  61
3a51 : ce fc 07 28 f7 df 4e 31  7e
3a59 : d0 7f 8e e2 0b 49 5e e9  31
3a61 : a8 e4 d6 2b b4 61 02 38  65
3a69 : 46 32 1b dd 0a 45 d1 6b  34
3a71 : 73 69 ad a8 ed 96 a5 b5  af
3a79 : a3 46 d2 89 74 b8 39 26  63
3a81 : 6b 84 8f 24 22 fd 78 46  17
3a89 : 7b 60 31 45 4f db 01 91  24
3a91 : 3e 40 46 19 13 60 0c 96  36
3a99 : 03 13 94 e3 83 11 c1 6b  66
3aa1 : cc a9 0d c2 7a 8c 30 0b  c0
3aa9 : 5e 65 48 6e 1a d5 38 ef  ab
3ab1 : b8 98 c3 22 76 e0 37 a9  89
3ab9 : f2 02 05 ed 23 23 84 86  16
3ac1 : 02 7b ef f4 c6 90 c0 27  5d
3ac9 : 6e 10 2f 69 18 c0 48 8e  fe
3ad1 : 11 49 9d cd e1 65 91 1c  70
3ad9 : 44 ed c1 8b 70 5a f2 a0  dc
3ae1 : c6 e1 3d f2 74 8c b7 01  d2
3ae9 : 89 ca 71 c2 07 13 19 8b  11
3af1 : 70 5a f2 a0 c6 e1 ac fa  83
3af9 : f6 91 96 e0 31 45 4d 80  ed
3b01 : 81 c4 c6 8c 6a 28 11 bb  cb
3b09 : eb aa 51 86 24 ed c0 d4  cd
3b11 : 78 d5 1b ef 0e dc 0d 45  bf
3b19 : 8b 70 51 a0 be 0e c5 42  3d
3b21 : 6c 83 0c 86 e3 d0 97 e8  18
3b29 : 34 1d d3 ef 32 93 12 41  69
3b31 : ad 42 72 42 5d 9b 42 5d  5b
3b39 : 6e cd a5 2f d0 69 cd 75  58
3b41 : 32 48 63 be f3 69 1d e8  19
3b49 : 70 32 7b 70 36 94 b8 2d  04
3b51 : 39 bb 8d b9 de 87 d3 b7  eb
3b59 : 95 e6 73 5c 76 7b 66 f6  94
3b61 : 72 7b 71 90 de e8 7b 05  2c
3b69 : b5 a0 8b 59 e8 bb b4 9e  f9
3b71 : 5e 2d a7 7b 3f 1c bf 63  5a
3b79 : 83 92 66 b8 48 f2 42 2f  79
3b81 : d7 84 67 ca 03 14 54 fe  ee
3b89 : 50 68 4d c7 ba 6a 26 9e  2e
3b91 : 93 84 55 d8 60 d3 d0 10  df
3b99 : ed 77 6f 6c e5 fb ad 09  b2
3ba1 : 74 fb 62 93 41 6e 3d d8  4c
3ba9 : 1f 72 93 42 6e 3d d8 1f  a1
3bb1 : 72 91 a7 a0 25 ad 54 97  2a
3bb9 : ee b4 15 d5 4e 3b ea 8b  83
3bc1 : 38 1f 72 92 b4 9f c6 35  45
3bc9 : da 02 f0 9b a9 2d ab 4c  9f
3bd1 : 5b aa 9b 97 d9 d6 6d 36  d2
3bd9 : bb 86 b0 38 99 06 bb 40  44
3be1 : 48 37 7d bf b3 68 4d c3  57
3be9 : 68 1c 4c 83 5d a0 28 96  8b
3bf1 : bb be 4d 9b 47 bb 86 d0  e0
3bf9 : be cf b0 d6 8d d3 59 f3  6a
3c01 : fb d9 ed a2 a3 69 74 d6  bd
3c09 : 7a dc 2b e2 f8 56 3c 9f  8b
3c11 : 7a cf 9a e3 d3 37 d3 db  94
3c19 : a7 e1 81 9e 18 0c 53 d4  be
3c21 : 33 c3 01 d7 6b 54 e3 be  d7
3c29 : fc b6 68 4a d1 d1 d1 c9  6a
3c31 : 85 02 d2 ce 07 13 20 d7  7f
3c39 : 68 0a c2 d7 77 fe 5e c3  c2
3c41 : 41 6e 1b 7d 9e ec 54 6f  b1
3c49 : 68 a2 8d a5 d3 59 eb 16  fe
3c51 : 5e 76 6e 17 0b 2c be 15  a0
3c59 : f7 27 d2 b3 e6 b8 f4 cd  b2
3c61 : f4 f6 e9 e4 01 99 00 3f  43
3c69 : e8 9e a1 99 00 3f f5 d6  bb
3c71 : a9 c7 7d f8 6c d0 7b a3  ff
3c79 : a3 93 0a 06 d2 cd 8c 80  f8
3c81 : dc 7c f0 be cf 3d ac fc  43
3c89 : 74 6d 2e 9a cf 58 b2 f3  05
3c91 : bc ac 2e 16 59 7c 2b ee  f6
3c99 : 4e 55 ef 35 c7 a6 6f a7  f3
3ca1 : b7 4d 40 31 40 91 90 13  2e
3ca9 : d4 31 40 1f cd ad 53 8e  bf
3cb1 : f9 6b d4 25 68 e8 e8 e4  75
3cb9 : c2 81 f8 6c d8 50 37 1f  33
3cc1 : 44 0e 26 47 a6 42 00 00  fb
```

Listing. »Basic-Maker« (Schluß)

Garbage 64 Version 2

Auch Gutes kann noch verbessert werden. Diese schnelle Garbage Collection ist um 25 Prozent kürzer als die des Listings des Monats aus der Ausgabe 2/86. Sie korrigiert das bei der Version 1 auftretende Problem mit dem String-Stack und ist trotzdem nicht wesentlich langsamer.

Normalerweise behindert sie uns nicht, die Garbage Collection. Tritt sie aber doch einmal auf, kann die Ausführung mehr als eine Stunde dauern. Meistens passiert das nur bei größeren Datenmengen, da hier sowieso viel Zeit für die Bearbeitung von Daten benötigt wird. Murphy's Gesetze zeigen auch hier ihre Wirkung.

»Garbage 64 V2« ist vollständig überarbeitet, daher genügt es nicht, Korrekturen an Version 1 vorzunehmen. Tippen Sie das Listing bitte mit dem MSE ein (siehe Listing). Das Einbinden ins Betriebssystem geschieht mit »SYS 51356«. Beim Auslösen der Garbage Collection erscheint links unten ein weißer Stern, dessen Zeichen und Farbe Sie in den Speicherstellen 50953 (Zeichen) und 50958 (Farbwert) verändern. Insgesamt läuft die neue Garbage Collection mehr als 2500mal schneller ab als normal. (J. Varnholdt/og)

```
programm : garbage 64 v2    c700 c8b7
-------------------------------------

c700 : a5 01 85 60 a9 37 85 01  00
c708 : a9 2a 8d e7 07 a9 01 8d  03
c710 : e7 db a6 37 a5 38 86 59  5e
c718 : 85 5a a9 19 a0 00 85 22  bc
c720 : 84 23 84 5f 84 57 84 58  09
c728 : c5 16 f0 05 20 ab c7 f0  35
c730 : f7 a9 07 85 53 a5 2d a6  d3
c738 : 2e 85 22 86 23 e4 30 d0  3e
c740 : 04 c5 2f f0 05 20 a1 c7  78
c748 : f0 f3 85 4e 86 4f a9 03  ed
c750 : 85 53 a5 4e a6 4f e4 32  8f
c758 : d0 07 c5 31 d0 03 4c 13  c0
c760 : c8 85 22 86 23 a0 00 b1  df
c768 : 22 aa c8 b1 22 08 c8 b1  31
c770 : 22 65 4e 85 4e c8 b1 22  bf
c778 : 65 4f 85 4f 28 10 d3 8a  38
c780 : 30 d0 c8 b1 22 a0 00 0a  bc
c788 : 69 05 65 22 85 22 90 02  c1
c790 : e6 23 a6 23 e4 4f d0 04  2a
c798 : c5 4e f0 ba 20 ab c7 f0  78
c7a0 : f3 b1 22 30 5e c8 b1 22  32
c7a8 : 10 59 c8 b1 22 f0 54 85  d3
c7b0 : 5d c8 b1 22 aa c8 b1 22  1e
c7b8 : c5 34 90 47 d0 04 e4 33  cb
c7c0 : 90 41 86 5b 85 5c a5 57  7e
c7c8 : a6 58 e0 e0 d0 0c c5 5d  2e
c7d0 : b0 08 c6 5f 85 5e a9 00  14
c7d8 : a2 c0 38 e5 5d 85 57 b0  66
c7e0 : 01 ca 86 58 a5 59 a6 5a  67
c7e8 : 38 e5 5d 85 59 88 91 22  7f
c7f0 : b0 01 ca c8 8a 85 5a 91  4e
c7f8 : 22 a4 5d 88 b1 5b 91 57  c0
c800 : 98 d0 f8 a5 53 18 65 22  c3
c808 : 85 22 90 02 e6 23 a6 23  6b
c810 : a0 00 60 a5 57 a6 58 f0  6b
c818 : 30 85 5b 86 5c 78 a9 35  4d
c820 : 85 01 a5 59 a6 5a 85 33  74
c828 : 86 34 a5 5f 10 0d a2 bf  91
c830 : 20 54 c8 a5 5e a2 e0 85  eb
c838 : 5b 86 5c a2 ff 20 54 c8  26
c840 : a5 60 c9 37 f0 03 20 72  fb
c848 : c8 a9 20 8d e7 07 a5 60  ad
c850 : 85 01 58 60 a0 00 b1 5b  ff
c858 : 91 59 e6 59 d0 02 e6 5a  e8
c860 : 88 c4 5b d0 05 e4 5c d0  c6
c868 : 01 60 e6 5b d0 e6 e6 5c  57
c870 : d0 e2 a9 37 85 01 a9 e0  cb
c878 : 20 87 c8 a5 5f f0 05 a9  27
c880 : a0 20 87 c8 4c ab c8 a0  b2
c888 : 00 84 22 85 23 a2 20 b1  2f
c890 : 22 91 22 c8 d0 f9 e6 23  db
c898 : ca d0 f4 60 a9 ff 85 5f  83
c8a0 : 20 72 c8 a9 35 85 01 60  a5
c8a8 : 4c 00 c7 a0 02 b9 a8 c8  1c
c8b0 : 99 26 b5 88 10 f7 60 00  1d
```

Listing. »Garbage 64 V2«

Programmierhilfe für Basic-Programmierer

Dieses Toolkit unterstützt den Basic-Programmierer durch einige sehr nützliche Befehle, die ihm die Arbeit erleichtern.

Jeder Basic-Programmierer wünscht sich für seinen C 64 Befehle, die ihm das Arbeiten mit seinem Computer erleichtern. Diese Befehle werden allgemein als Toolkit bezeichnet. Sie dienen dazu, diverse Aufgaben, die sonst vom Programmierer erledigt werden müssen, automatisch, das heißt ohne Dazutun des Anwenders zu erledigen. Etwa die automatische Zeichennummervorgabe beim Schreiben eines Basic-Programms (AUTO), das Löschen von bestimmten Zeilenblöcken (DELETE) oder auch ein Verbinden von zwei Programmen zu einem einzigen, ablauffähigen Programm (MERGE), das von Hand nicht so einfach zu lösen wäre. Ebenfalls unverzichtbar ist auch der RENUMBER-Befehl, der dazu dient, die Zeilen eines Basic-Programms neu zu numerieren, um eventuell Platz für neu einzufügende Zeilen zu schaffen.

Praktisch wäre auch noch ein Befehl zum Anzeigen der momentan im Programm verwendeten Variablen sowie deren derzeitige Belegung (DUMP).

Dieser Toolkit stellt Ihnen diese und noch weitere zur Verfügung.

Die Befehle und ihre Funktion

RENUMBER – Neunumerierung eines Programms (ohne GOTO/GOSUB).
Syntax: RENUMBER X,Y

Parameter: X – Nummer der ersten Programmzeile
Y – Schrittweite

DELETE – Löschen von Zeilenblöcken.
Syntax: DELETE X-Y
Parameter: X/Y – Anfang/Ende des zu löschenden Bereiches.

MERGE – Verbinden zweier Basic-Programme.
Syntax: MERGE "NAME",8 (oder ,1)
Das erste Programm muß sich im Speicher befinden. Danach MERGEt man einfach das zweite hinzu (die Zeilennummern des nachgeladenen Programms müssen höher als die des ersten sein).

DUMP – Anzeige der momentan belegten Variablen.
Syntax: DUMP

CATALOG – Anzeige des Disketten-Directories ohne Verlust eines im Speicher befindlichen Programms.
Syntax: CATALOG

OLD – Wiederherstellen eines durch NEW oder Reset gelöschten Programms.
Syntax: OLD

AUTO – Automatische Zeilennummer-Vorgabe.
Syntax: AUTO X,Y
Parameter: X – Anfangszeilennummer
Y – Schrittweite
Beenden der AUTO-Funktion durch Drücken von <RETURN> bei leerer Zeilennummer.

LSAVE – Speichern von Zeilenbereichen.
Syntax: LSAVE X-Y
Parameter: X/Y – Anfang/Ende des zu speichernden Bereiches.
Dieser Befehl gibt Ihnen die Möglichkeit, Teile, das heißt bestimmte Zeilennummern eines Basic-Programms, auf Diskette oder Kassette zu speichern (zum Beispiel LSAVE 400-900).

TRACE – Überwachen eines Basic-Programms.
Syntax: TRACE
In der rechten oberen Ecke wird die gerade bearbeitete Zeilennummer des laufenden Programms angezeigt.

OFF – Abschalten der TRACE-Funktion.
Syntax: OFF

FIND – Suchen bestimmter Befehle oder Zeichenfolgen.
Syntax: FIND PRINT oder
FIND "Text"
Bei der ersten Syntax wird nach Token gesucht, die zweite findet unverschlüsselte Texte.

STEP – Schrittweises Abarbeiten eines Basic-Programms.
Syntax: STEP gefolgt von TRACE
Nach jedem abgearbeiteten Befehl stoppt das Programm und wartet auf einen Tastendruck, um den nächsten Befehl zu bearbeiten.

SHOW – Anzeige der gerade bearbeiteten Zeile.
Syntax: SHOW gefolgt von TRACE
Es wird nicht mehr die Zeilennummer, sondern die komplette Basic-Zeile beim Testlauf ausgegeben.

! – Umrechnung Dezimal-Hexadezimal.
Syntax: !WERT
Zum Beispiel: !36578 !4553
Ergebnis: $8EE2 $11C9

$ – Umrechnung Hexadezimal-Dezimal
Syntax: $WERT
Zum Beispiel: $CE00 $00BE
Ergebnis: 52736 190

% – Umrechnung Binär-Dezimal.
Syntax: %WERT
Zum Beispiel: %11001001 %10011
Ergebnis: 201 19

@ – Kommando an die Floppy senden.
Syntax: @ N: TEST,00
Der Klammeraffe ohne weitere Angaben liest den Fehlerkanal der Floppy und zeigt ihn an.

Außerdem ist noch eine weitere Funktion integriert: HELP. Tritt ein Fehler während eines Programmlaufs auf, so wird die entsprechende Zeile auf dem Bildschirm gelistet und die Stelle, an der der Fehler auftrat, revers angezeigt.

Des weiteren sind die Funktionstasten wie folgt mit nützlichen Befehlen belegt:
F1 – LIST <RETURN>
F2 – Verlassen des Toolkits (Neustart: SYS 49152)
F3 – RUN <RETURN>
F4 – SYS 4096*
F5 – LOAD
F6 – OLD <RETURN>
F7 – CATALOG <RETURN>
F8 – Disk-Status

Die Funktionstasten F1 und F3 sind im TRACE-Modus mit folgenden Funktionen belegt:
F1 – STEP/normaler TRACE
F3 – SHOW/normaler TRACE

Eingabehinweise

Das Programm (Listing 1) geben Sie bitte mit dem MSE ein und speichern es auf Ihrem Datenträger. Gestartet wird es mit RUN. Haben Sie das Programm mit F2 verlassen, ist ein Neustart über SYS 49152 möglich. (Thomas Meidinger/dm)

```
name : top-tool            0801 12d5
-------------------------------------
0801 : 0b 08 0a 00 9e 32 30 36   3c
0809 : 34 00 00 00 00 00 00 78   2e
0811 : a9 00 85 fe a9 c0 85 ff   b2
0819 : a9 38 85 fc a9 08 85 fd   cc
0821 : a0 00 b1 fc 91 fe c8 d0   a3
0829 : f9 e6 fd e6 ff a5 ff c9   b2
0831 : cb d0 ef 58 4c 00 c0 a9   86
0839 : 00 8d 34 03 8d 36 03 85   0f
0841 : 24 85 25 20 44 a6 20 8c   88
0849 : c0 a9 93 20 d2 ff a2 01   80
0851 : a0 08 20 0c e5 a9 cd a0   a3
0859 : c9 20 c5 c6 a2 02 a0 07   47
0861 : 20 0c e5 a9 d6 a0 c9 20   10
0869 : c5 c6 a2 03 a0 02 20 0c   4d
0871 : e5 a9 e1 a0 c9 20 c5 c6   f9
0879 : a5 24 8d a6 cb a5 25 8d   02
0881 : a7 cb a2 05 a0 01 20 0c   02
0889 : e5 20 30 e4 78 20 53 e4   c7
0891 : a2 0b a9 62 8d 08 03 a9   e8
0899 : c1 8d 09 03 a9 4b 8d 00   ef
08a1 : 03 a9 c1 8d 01 03 a0 05   50
08a9 : b9 6c ca 99 14 03 88 10   1a
08b1 : f7 bd 9c cb c9 0e d0 38   e8
08b9 : ca bd 9c cb c9 4f d0 30   bd
08c1 : 6c 02 a0 a9 00 8d 20 d0   1a
08c9 : 8d 21 d0 a9 0d 8d 86 02   ab
08d1 : 60 48 8a 48 98 48 a9 7f   72
08d9 : 8d 0d dd ac 0d dd 30 1c   b2
08e1 : 20 02 fd d0 03 6c 02 80   38
08e9 : 20 bc f6 20 e1 ff d0 0c   a2
08f1 : 20 a3 fd 20 18 e5 20 8c   b1
08f9 : c0 4c 55 c0 4c 72 fe 78   92
0901 : a5 9d f0 1a a5 c5 cd 35   1e
0909 : 03 f0 13 8d 35 03 c9 04   95
0911 : f0 0f c9 05 f0 2d c9 06   47
0919 : f0 39 c9 03 f0 45 4c 31   45
0921 : ea ad 8e 02 d0 07 a2 06   a2
0929 : a0 06 4c 3c c1 20 15 fd   d4
0931 : 20 53 e4 a0 00 b9 c5 c9   c0
0939 : 20 d2 ff c8 c0 08 d0 f5   57
0941 : 4c 31 ea a2 05 a0 0b ad   12
0949 : 8e 02 f0 04 a2 08 a0 13   a8
0951 : 4c 3c c1 a2 04 a0 17 ad   7d
0959 : 8e 02 f0 04 a2 04 a0 1b   a8
0961 : 4c 3c c1 a2 09 a0 24 ad   11
0969 : 8e 02 f0 04 a2 02 a0 26   be
0971 : 4c 3c c1 86 c6 b9 f5 c9   c2
0979 : 9d 76 02 88 ca d0 f6 4c   8b
0981 : bc fe 8a 30 11 a5 9d d0   bb
0989 : 0a a9 32 8d 02 03 a9 c9   19
0991 : 8d 03 03 4c 3a a4 4c 74   cd
```

Listing 1. »Top-Tool« – Bitte mit dem MSE eingeben.

```
0999 : a4 a5 9d f0 35 a6 7a a5  53
09a1 : 7b 86 fd 85 fe a2 00 a0  d6
09a9 : 00 20 73 00 d9 1c ca d0  e1
09b1 : 03 c8 d0 f5 a9 ff d9 1c  45
09b9 : ca f0 1f c8 d9 1c ca d0  27
09c1 : fa a5 fd 85 7a a5 fe 85  9a
09c9 : 7b c8 e8 e0 15 d0 da 4c  da
09d1 : e4 a7 ad 34 03 f0 f8 4c  af
09d9 : c3 c5 bd 72 ca 48 bd 87  31
09e1 : ca 48 60 20 1f c9 a6 14  ee
09e9 : a4 15 86 f9 84 fa 20 fd  95
09f1 : ae 20 1f c9 a6 14 a4 15  78
09f9 : 86 f7 84 f8 78 20 14 c2  19
0a01 : a6 2b a4 2c 86 fb 84 fc  40
0a09 : a0 00 b1 fb 85 fd c8 b1  64
0a11 : fb f0 32 85 fe 18 a5 fb  01
0a19 : 69 02 85 fb a5 fc 69 00  4c
0a21 : 85 fc a0 00 a5 f9 91 fb  b5
0a29 : c8 a5 fa 91 fb a5 fd 85  a4
0a31 : fb a5 fe 85 fc 18 a5 f9  8a
0a39 : 65 f7 85 f9 a5 fa 65 f8  f4
0a41 : 85 fa 4c d1 c1 20 59 a6  60
0a49 : 4c 86 e3 a6 2b a5 2c 86  44
0a51 : fd 85 fe a0 04 b1 fd f0  8c
0a59 : 11 10 0c c9 89 f0 16 c9  bb
0a61 : 8d f0 12 c9 a7 f0 0e c8  f0
0a69 : d0 eb a0 00 b1 fd aa c8  9e
0a71 : b1 fd d0 db 60 84 02 a6  50
0a79 : fe 18 98 65 fd 90 01 e8  90
0a81 : 85 7a 86 7b 20 73 00 90  13
0a89 : 03 4c 76 c3 a6 7a a4 7b  90
0a91 : 86 fb 84 fc 20 1f c9 a9  4b
0a99 : 00 85 3f 85 40 a5 2b a6  08
0aa1 : 2c a0 00 85 5f 86 60 b1  dd
0aa9 : 5f 85 61 c8 b1 5f aa f0  df
0ab1 : 18 c8 a5 14 d1 5f f0 0a  09
0ab9 : a5 61 e6 3f d0 e3 e6 40  f9
0ac1 : d0 df c8 a5 15 d1 5f d0  67
0ac9 : ef a5 f7 85 61 a5 f8 85  6c
0ad1 : 62 a9 00 85 63 85 64 a2  f2
0ad9 : 10 46 40 66 3f 90 0d 18  c6
0ae1 : a5 61 65 63 85 63 a5 62  cb
0ae9 : 65 64 85 64 06 61 26 62  37
0af1 : ca d0 e6 18 a5 f9 65 63  66
0af9 : aa a5 fa 65 64 20 25 c9  51
0b01 : a0 ff c8 b1 fb c9 30 90  f9
0b09 : 04 c9 3a 90 f5 84 3f 18  43
0b11 : 98 65 fb 85 7a a5 fc 69  a7
0b19 : 00 85 7b a0 ff c8 b9 00  fc
0b21 : 01 d0 fa 84 40 38 a5 3f  b4
0b29 : e5 40 b0 2a a6 2d a4 2e  62
0b31 : 86 5a 84 5b 38 a5 40 e5  ee
0b39 : 3f 18 65 2d 85 58 85 2d  0f
0b41 : a5 2e 69 00 85 59 85 2e  ed
0b49 : a5 fb 85 5f a5 fc 85 60  52
0b51 : 20 bf a3 4c 51 c3 85 3f  8b
0b59 : a5 7a 85 5f e5 3f 85 58  a8
0b61 : a5 7b 85 60 e9 00 85 59  99
0b69 : a0 00 b1 5f 91 58 c8 d0  02
0b71 : f9 e6 59 e6 60 a5 2e c5  88
0b79 : 60 b0 ef 38 a5 2d e5 3f  0e
0b81 : 85 2d a5 2e e9 00 85 2e  dd
0b89 : a0 00 b9 00 01 f0 05 91  67
0b91 : fb c8 d0 f6 18 98 65 fb  d7
0b99 : 85 7a a5 fc 69 00 85 7b  08
0ba1 : 20 33 a5 20 79 00 c9 2c  df
0ba9 : d0 03 4c 4d c2 a4 02 4c  aa
0bb1 : 30 c2 20 1f c9 20 17 c9  bc
0bb9 : a9 24 20 d2 ff a2 01 b5  5b
0bc1 : 14 4a 4a 4a 4a 20 a1 c3  8a
0bc9 : b5 14 29 0f 20 a1 c3 ca  68
0bd1 : 10 ed 20 1a c9 4c 86 e3  04
0bd9 : 09 30 c9 3a 90 02 69 06  7f
0be1 : 20 d2 ff 60 4c 08 af 20  7a
0be9 : f1 c3 a2 01 20 79 00 f0  34
0bf1 : f3 e9 2f c9 0a 90 02 e9  df
0bf9 : 08 0a 0a 0a 0a 95 14 20  a8
0c01 : 73 00 f0 14 e9 2f c9 0a  86
0c09 : 90 02 e9 08 15 14 95 14  86
0c11 : 20 73 00 f0 03 ca 10 d9  83
0c19 : 20 17 c9 a6 14 a5 15 20  0f
0c21 : cd bd 20 1a c9 4c 86 e3  f9
0c29 : a9 00 85 14 85 15 60 20  79
0c31 : f1 c3 20 79 00 f0 ad c9  0d
0c39 : 31 18 d0 01 38 26 14 26  1c
0c41 : 15 20 73 00 90 f1 b0 d0  40
0c49 : 4c e3 a8 20 79 00 20 6b  a4
0c51 : a9 20 13 a6 90 f2 a6 5f  9e
0c59 : a4 60 86 fd 84 fe 20 fd  4b
0c61 : ae 20 1f c9 20 13 a6 90  77
0c69 : df 78 a0 00 b1 5f aa c8  ff
0c71 : b1 5f 85 60 86 5f b1 5f  28
0c79 : 91 fd c8 d0 f9 e6 60 e6  7b
0c81 : fe a5 2e c5 60 b0 ef 20  22
0c89 : 33 a5 20 9b a6 4c 86 e3  b9
0c91 : 20 79 00 20 6b a9 a6 14  39
0c99 : a4 15 86 f9 84 fa 20 fd  45
0ca1 : ae 20 6b a9 a6 14 a4 15  37
0ca9 : 86 f7 84 f8 ad 04 03 85  7d
0cb1 : fb a9 93 8d 04 03 ad 05  30
0cb9 : 03 85 fc a9 c4 8d 05 03  c6
0cc1 : a6 f9 a5 fa 20 bc c4 6c  00
0cc9 : 02 03 68 68 20 7c a5 ad  4c
0cd1 : 00 02 d0 0d a5 fb 8d 04  21
0cd9 : 03 a5 fc 8d 05 03 6c 02  be
0ce1 : 03 84 0b 18 a5 14 65 f7  6d
0ce9 : aa a5 15 65 f8 20 bc c4  65
0cf1 : 4c a4 a4 20 25 c9 a0 ff  e0
0cf9 : c8 b9 00 01 f0 05 99 77  4b
0d01 : 02 d0 f5 a9 20 99 77 02  cf
0d09 : c8 84 c6 60 20 70 c5 20  ae
0d11 : 59 a6 a9 00 85 b9 38 a5  7a
0d19 : 2d e9 02 aa a5 2e e9 00  84
0d21 : a8 a9 00 20 d5 ff b0 03  c8
0d29 : 4c a7 e1 a2 04 4c 3a a4  ea
0d31 : 4c 11 c4 20 79 00 20 6b  2a
0d39 : a9 20 13 a6 90 f2 a6 5f  86
0d41 : a4 60 86 39 84 3a 20 fd  75
0d49 : ae 20 6b a9 20 13 a6 90  6e
0d51 : df a0 00 b1 5f aa c8 b1  88
0d59 : 5f 86 3b 85 3c a0 02 b1  af
0d61 : 3b 99 e8 07 a9 00 91 3b  db
0d69 : 88 10 f4 20 fd ae 20 70  f1
0d71 : c5 a5 3b a4 3c 18 69 02  9a
0d79 : aa 90 01 c8 a9 39 20 d8  5b
0d81 : ff 08 a0 02 b9 e8 07 91  0f
0d89 : 3b 88 10 f8 28 b0 03 4c  d8
0d91 : 86 e3 a2 07 4c 3a a4 a9  0f
0d99 : 01 0c a9 00 85 0a 20 70  15
0da1 : c5 20 6f e1 4c 86 e3 a9  6a
0da9 : 00 20 bd ff a2 08 a0 01  18
0db1 : 20 ba ff 20 06 e2 20 57  d9
0db9 : e2 20 79 00 c9 2c f0 01  cd
0dc1 : 60 4c e9 e1 20 70 c5 20  db
0dc9 : 59 e1 4c 86 e3 ad ff 02  a6
0dd1 : 49 ff 8d ff 02 4c 86 e3  e1
0dd9 : a9 00 85 9d a6 fd a4 fe  82
0de1 : 86 7a 84 7b 4c e4 a7 a9  13
0de9 : ff 8d 34 03 4c 86 e3 ad  00
0df1 : 36 03 49 ff 8d 36 03 4c  2a
0df9 : 86 e3 a5 3a c9 ff f0 6e  5f
0e01 : cd d6 ca d0 07 a5 39 cd  24
0e09 : d5 ca f0 62 a2 00 b5 00  cd
0e11 : 9d 9c ca e8 d0 f8 a2 05  35
0e19 : bd 00 01 9d 9c cb ca 10  3e
0e21 : f7 ad 36 03 d0 58 a5 3a  b8
0e29 : a6 39 20 25 c9 a0 04 a9  1e
0e31 : a0 99 23 04 88 d0 [illegible] 20  22
0e39 : 3a c6 a0 05 99 22 d8 88  be
0e41 : 10 fa a9 a3 8d 22 04 a0  e8
0e49 : ff c8 b9 00 01 f0 07 09  e0
0e51 : 80 99 23 04 d0 f3 a2 00  1e
0e59 : bd 9c ca 95 00 e8 d0 f8  46
0e61 : a2 05 bd 9c cb 9d 00 01  34
0e69 : ca 10 f7 20 75 c6 4c e4  c6
0e71 : a7 a2 00 ad 21 d0 29 0f  7a
0e79 : d0 02 a2 01 8a 60 a2 02  4d
0e81 : 20 ff e9 ca 10 fa 20 66  9b
0e89 : e5 a5 3a 85 15 a6 39 86  f9
0e91 : 14 20 cd bd 20 13 a6 a9  69
0e99 : 00 85 7a 20 62 c9 20 3a  68
0ea1 : c6 a0 77 99 00 d8 88 10  d1
0ea9 : fa 4c 1f c6 ad ff 02 f0  2f
0eb1 : 10 20 e4 ff f0 fb c9 85  2b
0eb9 : f0 14 c9 86 f0 24 4c a5  a3
0ec1 : c6 20 e4 ff f0 15 c9 86  bc
0ec9 : f0 18 c9 85 d0 0d 20 a5  2a
0ed1 : c6 ad ff 02 49 ff 8d ff  79
0ed9 : 02 d0 d6 60 20 e4 ff d0  d0
0ee1 : fb 60 20 a5 c6 ad 36 03  82
0ee9 : 49 ff 8d 36 03 60 a9 00  36
0ef1 : 8d ff 02 8d 34 03 8d 36  ae
0ef9 : 03 4c 86 e3 85 22 84 23  02
0f01 : a0 00 b1 22 f0 10 aa 65  57
0f09 : 24 85 24 90 02 e6 25 c8  88
0f11 : 8a 20 d2 ff d0 ec 60 a0  97
0f19 : 00 20 79 00 99 a7 02 f0  48
0f21 : 60 c9 22 f0 5f c8 20 73  b0
0f29 : 00 99 a7 02 f0 05 c8 c0  fc
0f31 : 14 d0 f3 a6 2b a5 2c 86  1d
0f39 : fb 85 fc a0 03 b1 fb 99  2b
0f41 : 37 03 88 10 f8 a0 03 a2  04
0f49 : ff e8 c8 bd a7 02 f0 0a  08
0f51 : d1 fb f0 f5 b1 fb f0 1c  12
0f59 : d0 ed ae 39 03 86 14 ad  03
0f61 : 3a 03 85 15 20 cd bd 20  c8
0f69 : 13 a6 a9 00 85 7a 20 62  ab
0f71 : c9 20 1a c9 20 e1 ff f0  fd
0f79 : 08 ae 37 03 ad 38 03 d0  51
0f81 : b6 4c 86 e3 20 a2 e3 a0  63
0f89 : 00 b1 7a c9 22 d0 02 a9  3e
0f91 : 00 99 a7 02 f0 05 c8 c0  64
0f99 : 14 d0 ee 4c fc c6 38 a5  8d
0fa1 : 2d e5 2f aa a5 2e e5 30  a6
0fa9 : d0 07 e0 07 b0 03 4c 86  77
0fb1 : e3 a6 2d a4 2e 86 41 84  ec
0fb9 : 42 20 d6 c7 a0 00 b1 41  0d
0fc1 : 09 7f 30 18 c8 b1 41 09  ca
0fc9 : 7f 30 22 a9 a0 20 e7 c7  58
0fd1 : 20 a0 af 20 dd bd 20 1e  ba
0fd9 : ab 4c 07 c8 a9 25 20 e7  99
0fe1 : c7 20 61 af 20 dd bd 20  2f
0fe9 : 1e ab 4c 07 c8 a9 24 20  7b
0ff1 : e7 c7 20 14 c9 a0 00 b1  4b
0ff9 : 64 aa c8 b1 64 85 22 c8  a7
1001 : b1 64 85 23 20 25 ab 20  c4
1009 : 14 c9 4c 07 c8 a0 00 b1  eb
1011 : 41 29 7f f0 08 20 d2 ff  b2
1019 : c8 c0 02 d0 f2 60 20 d2  34
1021 : ff a6 d6 a0 04 20 0c e5  7a
1029 : a9 3d 20 d2 ff 20 17 c9  c4
1031 : 18 a5 41 69 02 85 64 a5  c3
1039 : 42 69 00 85 65 60 20 1a  ef
1041 : c9 18 a5 41 69 07 aa a5  6d
1049 : 42 69 00 a8 c4 30 90 04  6d
1051 : e4 2f b0 08 20 e1 ff f0  ed
1059 : 03 4c 7e c7 4c 86 e3 a9  f7
1061 : 01 a2 08 a0 0f 20 ba ff  a6
1069 : a9 00 20 bd ff 20 c0 ff  d6
1071 : 20 79 00 f0 17 a2 01 20  37
1079 : c9 ff 20 79 00 a0 00 c8  10
1081 : 20 d2 ff b1 7a 30 1f d0  88
1089 : f6 20 cc ff a2 01 20 c6  03
1091 : ff 20 13 ee 20 d2 ff 24  24
1099 : 90 50 f6 20 cc ff a9 01  88
10a1 : 20 c3 ff 4c 86 e3 38 e9  68
10a9 : 7f aa 84 8b 20 ac c9 a4  e8
10b1 : 8b 29 7f d0 ca a9 24 85  60
10b9 : fb a9 fb 85 bb a9 00 85  4c
10c1 : bc a9 01 85 b7 a9 08 85  37
10c9 : ba a9 60 85 b9 20 d5 f3  fc
10d1 : a5 ba 20 b4 ff a5 b9 20  c6
10d9 : 96 ff a9 00 85 90 a0 03  3f
10e1 : 84 fb 20 13 ee 85 fc a4  26
10e9 : 90 d0 3a 20 13 ee a4 90  d0
10f1 : d0 33 a4 fb 88 d0 e9 a6  07
10f9 : fc 20 cd bd 20 17 c9 20  52
1101 : 13 ee a6 90 d0 1f aa f0  da
1109 : 05 20 d2 ff d0 f1 20 1a  24
1111 : c9 20 e4 ff f0 09 c9 03  a8
1119 : f0 0b 20 e4 ff f0 fb a0  ec
1121 : 02 c9 03 d0 bb 20 42 f6  96
1129 : a5 90 29 bf d0 03 4c 86  bc
1131 : e3 a2 04 4c 3a a4 a9 01  61
1139 : a8 91 2b 20 33 a5 8a 69  d6
1141 : 02 85 2d a5 23 20 55 a6  dc
1149 : 4c 86 e3 a9 22 0c a9 20  70
1151 : 0c a9 0d 4c d2 ff 20 8a  c1
1159 : ad 4c f7 b7 85 62 86 63  6d
1161 : a2 90 38 20 49 bc 4c df  c9
1169 : bd a6 3a e0 ff f0 1d 86  2d
1171 : 15 a5 39 85 14 20 c9 bd  3d
1179 : a2 ff 86 3a 20 13 a6 18  69
1181 : a5 7a e5 5f 85 7a 20 62  3a
1189 : c9 20 1a c9 a9 83 8d 02  13
1191 : 03 a9 a4 8d 03 03 6c 02  42
1199 : 03 a0 03 84 0b 84 8b a9  94
11a1 : 20 a4 8b 29 7f 20 47 ab  89
11a9 : c9 22 d0 06 a5 0b 49 ff  50
11b1 : 85 0b a9 00 85 c7 c8 c4  69
11b9 : 7a d0 02 84 c7 b1 5f d0  d6
11c1 : 01 60 10 e1 c9 ff f0 dd  4f
11c9 : 24 0b 30 d9 c9 cc 90 09  11
11d1 : 84 8b 20 fd af a4 8b d0  d2
11d9 : d9 e9 7e aa 84 8b 20 ac  1a
11e1 : c9 30 be a0 ff ca f0 08  b0
11e9 : c8 b9 9e a0 10 fa 30 f5  cf
11f1 : c8 b9 9e a0 30 05 20 47  8c
11f9 : ab d0 f5 60 53 55 49 43  21
1201 : 49 44 45 0d 54 4f 50 2d  bb
1209 : 54 4f 4f 4c 00 a3 a3 a3  55
1211 : a3 a3 a3 a3 a3 a3 a3 00  c9
1219 : 56 4f 4e 20 54 48 4f 4d  0e
1221 : 41 53 20 4d 45 49 44 49  00
1229 : 4e 47 45 52 00 93 4c 49  17
1231 : 53 54 0d 93 52 55 4e 0d  87
1239 : 53 59 53 34 30 39 36 2a  8e
1241 : 4c 4f 41 44 4f 4c 44 0d  90
1249 : 93 43 41 54 41 4c 4f 47  9b
1251 : 0d 40 0d 52 45 4e 55 4d  c3
1259 : 42 45 52 ff 21 ff 24 ff  75
1261 : 44 45 88 45 ff 41 55 a4  bb
1269 : ff 4d 45 52 47 45 ff 54  f2
1271 : 52 41 43 45 ff 25 ff 4f  a5
1279 : 46 46 ff 46 49 4e 44 ff  c3
1281 : 44 55 4d 50 ff 95 ff 40  fa
1289 : ff 43 41 54 41 bc ff 4f  9d
1291 : 4c 44 ff 4c 94 ff 94 ff  24
1299 : 93 ff a9 ff 9a ff 53 48  1e
12a1 : 4f 57 ff c8 c0 b9 c0 9a  c7
12a9 : c0 c1 c3 c3 c4 c4 c4 c5  c4
12b1 : c3 c6 c6 c7 c5 c8 c8 c8  d9
12b9 : c4 c5 c5 c5 c5 c5 ab 7a  b8
12c1 : af 13 58 d4 af f7 b6 df  00
12c9 : 66 5f 27 7d fe fb 8c 62  1f
12d1 : 95 a0 b7 00 ff 00 ff 00  a4
```

Listing 1. »Top-Tool« (Schluß)

Berechnung periodischer Dezimalbrüche

Wahrscheinlich standen Sie auch schon vor dem Problem, einen periodischen Dezimalbruch in einen echten Bruch umzuwandeln. Dieses Programm berechnet Ihnen den echten Bruch automatisch. Ungenauigkeiten treten nicht auf.

Das Programm ist so geschrieben, daß eine Fehlbedienung so gut wie ausgeschlossen ist. Das Anfangsbild zeigt, wie die periodischen Dezimalbrüche eingegeben werden. Neben dem eigentlichen Ergebnis werden zusätzlich noch die wichtigsten Rechenschritte angezeigt. Nehmen wir doch einmal das Beispiel aus dem Anfangsbild:

Zu berechnen ist der Dezimalbruch
$0.76\overline{351}$,

Einzugeben ist der Wert 0.76351 und die Länge der Periode = drei. Der Dezimalbruch wird nun wie folgt aufgespaltet:

$$0.76\overline{351} = \frac{76}{100} + \text{Rest}$$

wobei für den Rest gilt,

$$\text{Rest} = \sum_{K=0}^{\infty} \frac{351}{10^{5+3K}}$$

Dieses ist noch recht einfach zu verstehen. Daraus ergibt sich

$$= \frac{351}{10^5} \times \sum_{K=0}^{\infty} \frac{1}{10^{3K}}$$

dieses ist durch vollständige Iteration beweisbar.

$$= \frac{351}{100000} \times \frac{1}{0.999}$$

Somit gilt (siehe auch ersten Zwischenschritt beim Ausdruck):

$$0.76\overline{351} = \frac{76}{100} + \frac{351}{100000} \times \frac{1}{0.999}$$



Dies war der schwierigste Teil der Berechnung und erforderte zum ganzen Verständnis schon einige Grundkenntnisse in höherer Mathematik. Doch zum Glück ist das Verständnis keine Voraussetzung zum Benutzen des Programms.

Sehen wir weiter: Zunächst wird nun versucht, die vorhandenen Brüche so klein wie möglich zu machen. Das geschieht dadurch, daß man jeweils von Zähler und Nenner den GGT (größter gemeinsamer Teiler) berechnet und Zähler und Nenner dadurch dividiert. In unserem Beispiel ist beim ersten Bruch

$$\frac{76}{100}$$

der GGT = 4; nach der Division erhalten wir also

$$\frac{19}{25}$$

Beim anderen Bruch (der zweite Bruch wurde bereits mit dem dritten multipliziert) haben wir den GGT 27, dies führt dann zum Ergebnis

$$\frac{13}{3700}$$

Als zweites Zwischenergebnis steht nun:

$$= \frac{19}{25} + \frac{13}{3700}$$

Im folgenden sind jetzt also nur noch die beiden Brüche zu addieren. Das machen wir, wie in der Schule gelernt, indem wir zuerst das KGV (kleinstes gemeinsames Vielfaches) der beiden Nenner suchen. Da die Funktion GGT schon vorhanden ist und das KGV definiert ist als

$$\text{KGV}(a,b) = \frac{a \times b}{\text{GGT}(a,b)}$$

ist dies kein größeres Problem.

Anschließend muß dann nochmal durch den GGT des jetzt so entstehenden einzigen Bruches dividiert werden und wir erhalten:

$$= \frac{113}{148}$$

Wenn das Programm als Unterprogramm verwendet werden soll, so muß vor dem Aufruf der Dezimalbruch in X$ stehen und die Länge in L$. Zurückgegeben wird dann in Z2$ der Zähler und in N2$ der Nenner. Die Zeilen zum Ausdrucken der Ergebnisse müssen natürlich entfernt werden.

Das Programm wurde zwar für den C 64 geschrieben, kann aber im Prinzip auf jeden anderen Computer übertragen werden, sofern er Stringbefehle verarbeiten kann. Falls diese nicht zur Verfügung stehen, müssen wenige Anpassungs-Routinen geschrieben werden. (B. Filpe/do)

```
0 REM ***********************************   <131>
1 REM *                                 *   <050>
2 REM *     BERECHNUNG PERIODISCHER     *   <145>
3 REM *         DEZIMALBRUECHE          *   <005>
4 REM *                                 *   <053>
5 REM *     BERNHARD FILPE              *   <177>
6 REM *     LANGGASSE 28                *   <036>
7 REM *     6733 HASSLOCH               *   <192>
8 REM *                                 *   <057>
9 REM ***********************************   <140>
10 POKE 53280,6:POKE 53281,6:PRINT CHR$(14
   7),CHR$(14),CHR$(8)                      <201>
20 PRINT"BERECHNUNG PERIODISCHER DEZIMALBR
   UECHE"                                   <062>
30 PRINT"TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
   TTTTT"                                   <094>
40 PRINT"{3DOWN}EINGABEBEISPIEL:"           <248>
45 PRINT"{DOWN,18SPACE}***"                 <127>
50 PRNT"ZU BERECNEN .7635"          <232>
60 PRINT"{DOWN}EINGABE{5SPACE}: 0.76351{3S
   PACE}LAENGE 3"                           <211>
70 PRINT"{DOWN}"                            <018>
80 INPUT"DEZIMALBRUCH{6SPACE}: ";X$         <232>
90 INPUT"LAENGE DER PERIODE: ";L$           <211>
100 L=VAL(L$):IF L<>INT(L)OR L<1 THEN ER$=
    "EINGABE DER PERIODE FALSCH":GOTO 2000 <005>
105 IF VAL(X$)=0 THEN ER$="FALSCHE EINGABE
    ":GOTO 2000                             <096>
110 REM TEST AUF KORREKTE EINGABE           <109>
120 XL=LEN(X$)                              <138>
130 FOR I=1 TO XL                           <196>
140 T$=MID$(X$,I,1):T=VAL(T$)               <144>
150 IF(T=>1)AND(T<=9)THEN FL=1:GOTO 180     <198>
160 IF T$="0"AND FL=0 THEN NU=NU+1:GOTO 18
    0                                       <005>
165 IF T$="0"THEN 180                       <166>
170 IF T$="."AND PF=0 THEN FL=1:PF=1:PK=I:
    GOTO 180                                <066>
175 ER$="FALSCHE EINGABE":EL=I:GOTO 2000    <238>
180 NEXT                                    <190>
200 IF FL THEN X$=MID$(X$,NU+1):PK=PK-NU    <253>
210 XL=LEN(X$)                              <228>
220 IF PF=0 THEN ER$="DEZIMALPUNKT FEHLT":
    GOTO 2000                               <247>
230 IF L>XL-PK THEN ER$="FALSCHE PERIODENA
    NGABE":GOTO 2000                        <248>
250 PRINT"{CLR,DOWN}BERECHNUNG PERIODISCHE
    R DEZIMALBRUECHE"                       <198>
260 PRINT"TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
    TTTTTT"                                 <070>
270 PRINT"{DOWN}DEZIMALBRUCH{7SPACE}: ";:I
    F PK=1 THEN PRINT"0";                   <017>
275 PRINT X$                                <127>
280 PRINT"LAENGE DER PERIODE : ";L$         <227>
290 PRINT                                   <138>
300 REM BERECHNUNG                          <045>
310 N(1)=10↑(XL-L-PK)                       <200>
320 IF PK=1 THEN V$=MID$(X$,2,XL-L-1):GOTO
    337                                     <012>
330 V$=MID$(X$,1,PK-1):VL=LEN(V$)           <183>
335 V$=V$+MID$(X$,PK+1,XL-VL-1-L)           <103>
337 Z(1)=VAL(V$)                            <112>
340 N(2)=10↑(XL-PK)                         <102>
350 Z(2)=VAL(MID$(X$,XL-L+1)):IF Z(2)=0 TH
    EN ER$="DEZIMALBRUCH TRIVIAL":GOTO 200
    0                                       <191>
360 N(3)=1-10↑(-L):Z(3)=1                   <187>
400 GOSUB 3000                              <118>
410 GOSUB 4000:PRINT                        <062>
420 IF Z(1)=0 THEN 450                      <236>
430 G1=Z(1):G2=N(1):GOSUB 5000:REM GGT      <217>
440 Z(1)=Z(1)/GG:N(1)=N(1)/GG               <005>
450 Z(2)=Z(2)*Z(3)                          <046>
460 N(2)=N(2)*N(3)                          <065>
510 G1=Z(2):G2=N(2):GOSUB 5000:REM GGT      <043>
520 Z(2)=Z(2)/GG:N(2)=N(2)/GG               <171>
530 GOSUB 3000                              <250>
540 IF Z(1)<>0 THEN GOSUB 4300              <211>
550 IF Z(1)=0 THEN GOSUB 4200:END           <157>
555 PRINT                                   <149>
560 G1=N(1):G2=N(2):GOSUB 5000              <099>
570 GG=N(1)*N(2)/GG:REM KGV                 <014>
580 Z(1)=Z(1)*GG/N(1)                       <178>
590 Z(2)=Z(2)*GG/N(2)                       <208>
600 N(2)=GG:Z(2)=Z(1)+Z(2)                  <064>
610 G1=Z(2):G2=N(2):GOSUB 5000              <152>
620 Z(2)=Z(2)/GG:N(2)=N(2)/GG               <015>
630 GOSUB 3000:GOSUB 4200                   <188>
700 END                                     <194>
2000 REM FEHLERANZEIGE                      <047>
2010 PRINT"{3DOWN,RVSON,SPACE}";ER$;" "     <096>
2015 IF EL=0 THEN 2030                      <029>
2017 PRINT"{DOWN}";X$:IF EL=1 THEN PRINT"↑
     ":GOTO 2030                            <080>
2020 FOR J=1 TO I-1:PRINT" ";:NEXT:PRINT"↑
     "                                      <016>
2030 FOR W=1 TO 1000:GET A$:IF A$=""THEN N
     EXT                                    <025>
2040 RUN                                    <018>
3000 REM AUSGABE DER ERRECHNETEN WERTE      <171>
3005 Z(1)=INT(Z(1)):Z(2)=INT(Z(2)):N(1)=IN
     T(N(1)):N(2)=INT(N(2))                 <169>
3010 Z1$=STR$(Z(1)):N1$=STR$(N(1))          <049>
3020 Z2$=STR$(Z(2)):N2$=STR$(N(2))          <008>
3030 Z3$="1":N3$=STR$(N(3)):IF N(3)<>1 THE
     N 3040                                 <106>
3035 N3$=" .":FOR J=1 TO L:N3$=N3$+"9":NEX
     T:N(3)=VAL(N3$)                        <114>
3040 L1=LEN(Z1$):IF L1<LEN(N1$)THEN L1=LEN
     (N1$)                                  <182>
3050 L2=LEN(Z2$):IF L2<LEN(N2$)THEN L2=LEN
     (N2$)                                  <050>
3060 L3=LEN(N3$)                            <199>
3070 IF L1>LEN(Z1$)THEN Z1$=" "+Z1$:GOTO 3
     070                                    <216>
3080 IF L1>LEN(N1$)THEN N1$=" "+N1$:GOTO 3
     080                                    <157>
3090 IF L2>LEN(Z2$)THEN Z2$=" "+Z2$:GOTO 3
     090                                    <112>
3100 IF L2>LEN(N2$)THEN N2$=" "+N2$:GOTO 3
     100                                    <231>
3110 IF L3>LEN(Z3$)THEN Z3$=" "+Z3$:GOTO 3
     110                                    <186>
3120 N3$="0"+MID$(N3$,2)                    <034>
3130 RETURN                                 <140>
4000 PRINT"{2SPACE}";Z1$;"{3SPACE}";Z2$;"{
     3SPACE}";Z3$                           <175>
4010 PRINT"= ";                             <036>
4020 FOR I=1 TO L1:PRINT"*";:NEXT:PRINT" +
     ";:FOR I=1 TO L2:PRINT"*";:NEXT:PRIN
     T" * ";                                <058>
4030 FOR I=1 TO L3:PRINT"*";:NEXT:PRINT     <129>
4040 PRINT"{2SPACE}";N1$;"{3SPACE}";N2$;"{
     3SPACE}";N3$                           <145>
4050 RETURN                                 <042>
4100 PRINT"{2SPACE}";Z2$;"{3SPACE}";Z3$     <226>
4110 PRINT"= ";                             <138>
4120 FOR I=1 TO L2:PRINT"*";:NEXT:PRINT" *
     ";                                     <161>
4130 FOR I=1 TO L3:PRINT"*";:NEXT:PRINT     <231>
4140 PRINT"{2SPACE}";N2$;"{3SPACE}";N3$     <133>
4150 RETURN                                 <144>
4200 PRINT"{2SPACE}";Z2$                    <091>
4210 PRINT"= ";                             <238>
4230 FOR I=1 TO L2:PRINT"*";:NEXT:PRINT     <074>
4240 PRINT"{2SPACE}";N2$                    <128>
4250 RETURN                                 <244>
4300 PRINT"{2SPACE}";Z1$;"{3SPACE}";Z2$     <233>
4310 PRINT"= ";                             <082>
4320 FOR I=1 TO L1:PRINT"*";:NEXT:PRINT" +
     ";                                     <120>
4330 FOR I=1 TO L2:PRINT"*";:NEXT:PRINT     <174>
4340 PRINT"{2SPACE}";N1$;"{3SPACE}";N2$     <140>
4350 RETURN                                 <088>
5000 REM GGT                                <036>
5010 G1=INT(G1):G2=INT(G2)                  <218>
5020 IF G1<G2 THEN HI=G1:G1=G2:G2=HI        <231>
5030 IF G1/G2=INT(G1/G2)THEN GG=G2:RETURN   <168>
5040 Q=G1/G2:IF Q>20 THEN G1=G1-INT(Q-10)*
     G2                                     <170>
5050 IF G1=G2 THEN GG=G1:RETURN             <241>
5060 IF G1>G2 THEN G1=INT(G1-G2):GOTO 5020 <241>
5070 G2=INT(G2-G1):GOTO 5020                <136>
```

Listing. Wandlungen eines periodischen Bruches in einen echten Bruch

Das erste Lebenszeichen

Was passiert in den ersten Sekunden nach dem Einschalten Ihres C64? Woher weiß der Prozessor, was er zu tun hat? Was ist ein Reset-Taster, was ein Modul-Start? Auf diese Fragen finden Sie auf dieser Seite ausführlich Antwort.

Haben Sie sich nicht auch schon Gedanken darüber gemacht, warum sich Ihr C64 nach dem Einschalten für kurze Zeit »totstellt«? Brauchen die Chips erst eine kurze Vorwärmzeit? Muß der Prozessor erst einmal wach werden?

Nein, während dieser Phase, in der der Computer zwar eingeschaltet ist, aber noch kein Lebenszeichen (sprich »READY.«-Meldung) von sich gibt, wird ein sogenannter »Reset« ausgeführt. Dieser Reset ist für den C64 nicht eine »Anwärmphase«, sondern eine höchst arbeitsintensive und interessante Routine. Schauen wir uns einmal an, was in unserem C64 nach dem Einschalten so alles vor sich geht.

Wenn Sie Ihren C64 einschalten, so werden die Bausteine im Computer nahezu schlagartig mit Spannung versorgt. Würde die CPU sofort losstürmen, würde sie sofort wieder in den Tiefen des C64 verschwinden. Daran hindert sie jedoch ein Signal am Reset-Eingang. Er liegt auf logisch »0« und lähmt somit den Prozessor. Zu diesem Zeitpunkt wird ein spezieller Timer-Baustein in Gang gesetzt, der die Reset-Leitung kontrolliert. Er wartet, bis sich die Betriebsspannung vollständig aufgebaut und stabilisiert hat. Erst nach Ablauf der hardwaremäßig vorgegebenen Zeitspanne wechselt der Timer seinen Pegel und gibt somit den Prozessor frei, der bis dahin noch nicht eine einzige Operation gemacht hat.

Doch dieser Reset (es gibt noch andere) betrifft nicht nur die CPU. Drei weitere Bausteine bekommen ihn auf die gleiche Weise zu spüren: der Sound-Chip (welch ein Lärm ohne Reset) und die zwei Port-Bausteine (CIAs). Primär betrifft der Einschalt-Reset also vier Bausteine. Doch nur beim Prozessor geht es aktiv weiter, die anderen drei werden nur in eine bestimmte Grundposition »gefahren«, um den Prozessor und uns nicht weiter zu stören.

Am Anfang kommt ein Reset

Wichtig ist, daß der Prozessor bei seinem ersten Zugriff das ROM vorfindet, denn nach Freigabe der Reset-Leitung führt die CPU einen bestimmten Befehl aus. Dafür sorgt der Prozessor selber, indem er an seinen Ausgangsports Basic, Kernel und I/O-Bereich selektiert. Der erste Befehl ist ein indirekter Sprung nach $FFFC, fast an das Ende des Speicherbereichs. Das heißt, der Prozessor »sieht nach«, welche 2-Byte-Adresse sich dort befindet, um sofort dorthin zu springen. Beim C64 steht dort die Adresse $FCE2 (64738). Und dann geht es richtig los.

Zunächst wird der Prozessor gegen normale Interrupts abgesichert (Einen NMI, also RUN/STOP RESTORE, nimmt er nach wie vor an, stürzt aber dabei ab.), der Stack-Pointer wird zurückgesetzt und das Dezimal-Flag (für uns nicht so wichtig, es ist hier nur der Vollständigkeit halber erwähnt) wird gelöscht. Dann prüft der Computer, ob sich ein Modul im Expansions-Port befindet. Diese Routine befindet sich ab $FD02. Er vergleicht die Adressen $8004 bis $8008, ob deren Inhalt den Speicherstellen $FD10 bis $FD14 entspricht. Sie enthalten die Zeichen »CBM80«.

Läuft der Modul-Vergleich erfolgreich ab, wird wieder ein indirekter Sprung ausgeführt, diesmal nach $8000. Es wird also auf die dort vorgefundene Adresse verzweigt. Allerdings erkennt der Computer nicht das Modul, sondern lediglich die Meldung »CBM80«. So ist es möglich, einen solchen Modul-Start zu simulieren, indem man die Modul-Identifikation einfach an die entsprechende Adresse (ab $8004) schreibt und den Vektor bei $8000 auf eine eigene Routine verzweigt. Bei professionellen Spielen wird auf diese Weise der Reset unterdrückt. Dafür bieten einige Speeder wie zum Beispiel SpeedDos, PrologicDos, DolphinDos und das 64'er-Dos die Möglichkeit, während des Resets durch Drücken einer Taste (<SPACE> oder <CTRL>) den Modulstart zu unterbinden. Dies ist allerdings eine Veränderung der Systemsoftware, die im Commodore-Betriebssystem nicht besteht.

Findet der Computer keinen Modul-Start vor, dann wird als nächstes das Register 22 des VIC auf Null gesetzt – das sehen Sie, wenn Sie einen Reset-Taster haben, am Abschneiden des rechten und linken Bildrandes.

Reset sorgt für Ruhe und Ordnung

Im darauffolgenden Schritt, einem Unterprogramm ab $FDA3, werden die Portbausteine und der Sound-Chip, die ja auch einen Reset durchgeführt haben, abgehandelt. Bei letzterem wird nur die Lautstärke auf Null gesetzt, für den Fall, daß der Reset softwaremäßig ausgelöst wurde. In den CIAs dagegen werden die Timer für den Interrupt vorbereitet und gesetzt, die Datenrichtungs-Register, die CIA-Ports und sowie der Prozessor-Port definiert.

Jetzt erst kommt der C64 dazu, seinen Arbeitsspeicher zu testen und zu initialisieren. Die Routine dazu steht ab $FD50 und ist zu etwa 98 Prozent verantwortlich für die Dauer des Resets. Der Bereich von $0002 bis $03FF wird komplett gelöscht, der gesamte Speicher von $0400 bis $9FFF wird durch zweimaliges Schreiben und Auslesen auf Funktion getestet. Daraufhin werden die Ober- und Untergrenze des Basic-Bereiches definiert und das Video-RAM (der Bildschirm) nach $0400 gelegt.

Hardware und Ein-/Ausgabe-Vektoren sind die nächste Station des Resets. Damit sind die Zeiger ab $0314 gemeint. Deren Inhalte werden von einer Routine ab $FD15 einfach aus einer Tabelle ab $FD30 kopiert.

Die letzte Station des Resets ist die Initialisierung des Video-Controllers. Eine Tabelle ab $ECB9 wird nach $D000 kopiert. Dadurch wird der VIC initialisiert. Danach wird die Tastatur-Eingabe vorbereitet, der Bildschirm gelöscht und nochmals der Interrupt-Timer gesetzt. Nachdem dann der Interrupt schließlich auch noch zugelassen wurde, wird der Basic-Kaltstart, durch einen indirekten Sprung nach $A000, ausgeführt.

Wir haben vorher angesprochen, daß es mehrere Möglichkeiten gibt, einen Reset auszulösen. Den Einschalt-Reset können Sie auslösen, indem Sie die Reset-Leitung kurzzeitig auf Masse ziehen, beispielsweise durch einen Taster. Eine Bauanleitung dafür finden Sie im 64'er, Ausgabe 6/85.

Einige Erweiterungen beinhalten Reset-Befehle, wie COLD bei Simons Basic, die dann einen Reset softwaremäßig auslösen. Dem entspricht ein SYS 64738 im normalen Basic. Allerdings ist dies kein »sauberer« Reset, da nicht automatisch auf das ROM geschaltet wird und auch der SID und die CIAs nicht davon betroffen sind. Der Einsprung ist allerdings derselbe, also mit Modul-Test und Initialisierung aller wichtigen Register, so daß Sie hier beliebig experimentieren können. Natürlich lassen sich die diversen Einsprünge auch unabhängig voneinander verwenden. So löst zum Beispiel SYS 64763 nur einen Video-Reset mit Basic-Kaltstart aus, ohne eventuell verstellte Interrupt-Vektoren zu korrigieren.

(og)

Grund genug fürs neue 64'er:



Mitmachen! Mitgewinnen!
Zwei große Umfrage-Aktionen in dieser Ausgabe:
1. *»Schule«–20 Brother-Drucker zu gewinnen!*
2. *»Ihr Wunschdrucker« – 24 Spitzendrucker winken als Preise!*

In der August-Ausgabe berichten wir über

Lernen mit dem Computer in Schule und Ausbildung

- In welchen Bereichen wird der Computer eingesetzt?
- Was wird mit ihm erarbeitet?
- Wie kann man mit ihm leichter lernen?
- Welchen Sinn haben Lernprogramme?

... außerdem lesen Sie:

■ Programmiersprache »Comal 80«, eine Kombination aus Basic und Pascal. Sehr komfortabel und einfach zu programmieren. Mit vielen Sound- und Grafikbefehlen. ■ 64'er-Extra mit Befehlsvergleichstabelle zwischen C64-, C128- und Simons-Basic sowie Pascal und Comal 80. ■ Hardware-Test: Drucker »Epson EX-800« und »Brother M 1009« sowie Floppyspeeder »Professional DOS«. ■ Reparatur-Kurs: »Jetzt helfe ich mir selbst«. ■ Anwendung des Monats: »Digitracer«, das Testprogramm für digitale Schaltungen.

Falls Sie »64'er« noch nicht regelmäßig beziehen, sichern Sie sich jetzt Ihr persönliches Abonnement und nutzen die damit verbundenen Vorteile: ■ Sie beziehen »64'er« ohne Mehrkosten bequem per Post frei Haus ■ Sie haben Ihr »64'er« bereits bei sich zu Hause — noch bevor Sie es bei Ihrem Zeitschriftenhändler kaufen können. ■ Sie sind sicher, keine Ausgabe zu versäumen.

Sie erhalten – wenn Sie zur Anforderung den nebenstehenden Gutschein verwenden — auf alle Fälle die neueste Ausgabe als Probeheft unverbindlich und kostenlos.

Gutschein

FÜR EIN KOSTENLOSES PROBEEXEMPLAR DES 64'er-MAGAZINS

JA, ich möchte »64'er«, das Magazin für Computerfans, kennenlernen. Senden Sie mir bitte die aktuellste Ausgabe kostenlos als Probeexemplar. Wenn mir »64'er« gefällt und ich es regelmäßig weiterbeziehen möchte, brauche ich nichts zu tun: Ich erhalte »64'er« dann regelmäßig frei Haus per Post und bezahle pro Jahr nur DM 78,— (Ausland auf Anfrage).

Vorname, Name

Straße PLZ, Ort

Datum 1. Unterschrift

Mir ist bekannt, daß ich diese Bestellung innerhalb von 8 Tagen bei der Bestelladresse widerrufen kann und bestätige dies durch meine zweite Unterschrift. Zur Wahrung der Frist genügt die rechtzeitige Absendung des Widerrufs.

Datum 2. Unterschrift

Gutschein ausfüllen, ausschneiden, in ein Kuvert stecken und absenden an:
Markt&Technik Verlag Aktiengesellschaft, Vertrieb, Postfach 1304, 8013 Haar

64S786

Label im Basic 7.0

Nun wird es auch mit Basic 7.0 möglich, Label anstatt feststehender Zeilennummern anzugeben. Dies erleichtert die Programmierung.

Obwohl das Basic 7.0 des C128 sehr umfangreich ist, fehlt ihm eine Möglichkeit, die einige andere Computer schon aufweisen: die Angabe von Sprungzielen durch Label.

Eine große Schwierigkeit beim Erstellen umfangreicher Programme scheint es zu sein, daß man von Anfang an wissen muß, wohin ein Sprung führen soll. Auch die Zusammenstellung von Programmen durch Module ist nicht einfach, wenn Sprungadressen auftreten.

Mit diesem Programm (Listing 1) kann neben der bisherigen Möglichkeit, Sprungziele mit Zeilennummern anzugeben, folgendes programmiert werden:

1) GOTO und GOSUB auf eine errechnete Zeile.

```
A=1000:GOTO A
A=1000:GOSUB A/2+50
```

Damit lassen sich Sprungverteiler vermeiden.

2) GOTO und GOSUB auf Inhalte von Feldelementen (numerisch). In einem numerischen Feld sind Sprungziele gespeichert. Hiermit ist möglich: GOTO (oder GOSUB) A(I) mit I aus zulässigem Bereich.

3) GOTO und GOSUB auf Marken. Die Marke muß in einer REM-Zeile stehen. Es ist folgendes möglich:

```
...
...
550 GOTO"DRUCK"
...
...
... REMDRUCK
... PRINT"...
```

Gleiches gilt auch für GOSUB.

```
name : sys                  1300 13b7
--------------------------------------
1300 : ad 00 ff 48 a9 00 8d 00   87
1308 : ff ad 06 d5 09 06 8d 06   1d
1310 : d5 68 8d 00 ff 78 a9 22   2b
1318 : 8d 08 03 a9 13 8d 09 03   67
1320 : 58 60 20 80 03 c9 89 f0   47
1328 : 0a c9 8d f0 24 4c f3 4a   a1
1330 : ea ea ea 20 f7 77 a5 0f   3e
1338 : 30 23 20 15 88 a5 17 c9   4a
1340 : fa b0 06 20 e2 59 4c f6   30
1348 : 4a a2 0b 2c a2 11 4c 3c   88
1350 : 4d 20 1d 5a 20 86 03 20   c2
1358 : 33'13 4c f6 4a 20 7e 87   b5
1360 : f0 ea 18 69 05 90 03 4c   72
1368 : ed a5 85 93 8a e9 04 85   0f
1370 : 5c 98 e9 00 85 5d a5 2d   c7
1378 : a6 2e 85 61 86 62 a0 01   c3
1380 : 20 ec 42 f0 c7 aa a0 04   21
1388 : 20 ec 42 c9 8f f0 14 a0   fa
1390 : 00 20 ec 42 4c 7a 13 20   49
1398 : e2 42 85 a6 20 ec 42 c5   cf
13a0 : a6 d0 ec c8 c4 93 d0 ef   0e
13a8 : 18 a0 01 20 94 50 86 62   ff
13b0 : 38 20 06 5a 4c f6 4a 18   9b
```

Listing 1. Bitte mit dem MSE im C64-Modus eingeben.

4) GOTO und GOSUB auf den Inhalt von Feldelementen, die Strings enthalten. Verfahren wie unter Punkt 2).

5) GOTO und GOSUB auf definierte Stringvariable. Dabei muß stets eine REM-Zeile mit dem entsprechenden String vorhanden sein.

```
10 A$="1000"
20 GOSUB A$
```

Eingabehinweise

Das Programm (Listing 1) geben sie bitte mit dem MSE im C64-Modus ein und speichern es. Für Floppy-Besitzer scheint weiter folgender Weg der einfachste zu sein:

1) Erstellen einer Autoboot-Diskette
Name des Basic-Programms: INIT

2) Unter INIT wird folgender Einzeiler gespeichert:

```
10 POKE 46,32:POKE 48,32:POKEDEC("2000"),0:CLR:
BLOAD"SYS":SYSDEC("1300"):NEW
```

Dieses Programm bewirkt folgendes:

a) Basic-Anfang auf $2000 einstellen
b) Variablen-Anfang auf $2000 einstellen
c) Der Befehl CLR bewirkt ein Anpassen der übrigen Zeiger auf den umgestellten Bereich
d) Nachladen des unter SYS gespeicherten Maschinenprogrammes
e) Einbinden in den Interpreter
f) Löschen

Statt f) könnte auch ein weiteres Programm nachgeladen und gestartet werden.

3) Das Maschinenprogramm ist dabei unter dem Namen SYS gespeichert.

Nicht-Floppy-Besitzer müssen folgenden Einzeiler laden und starten:

```
10 POKE 46,32:POKE 48,32: POKEDEC("2000"),0:CLR:NEW
```

Danach muß das Maschinenprogramm geladen und mit SYSDEC("1300") gestartet werden.

Die Listings 2 und 3 sollen Ihnen nur die Fähigkeiten der Erweiterung demonstrieren, müssen also nicht unbedingt abgetippt werden.

Um das Maschinenprogramm kurz zu halten, mußten folgende Restriktionen vereinbart werden:

1) Die erweiterte GOTO/GOSUB-Routine ist nur im Programm-Modus benutzbar

```
..:GOTO..
```

2) Soll nach THEN auf eine Marke gesprungen werden, muß die Syntax geändert werden

```
IF..THEN:GOTO..
```

3) Soll ELSE mit Marken verwendet werden, ist ebenfalls die Syntax zu ändern

```
IF..:ELSE:GOTO..
```

(Gerhard Seltzsam/dm)

```
10 REM    *** DEMO 1 ***                <A99>
15 PRINT CHR$(147)                      <PA3>
20 FOR I = 1 TO 10                      <I0V>
30 GOSUB "ZAHL"                         <510>
40 GOSUB "QUADRAT"                      <TUJ>
50 GOSUB "WURZEL"                       <QOP>
60 NEXT                                 <8P0>
70 END                                  <7SU>
100 REM ZAHL                            <IIA>
110 PRINT I,                            <3H7>
120 RETURN                              <4S7>
300 REM WURZEL                          <JIE>
310 PRINT SQR(I)                        <JJV>
320 RETURN                              <4K7>
510 REM QUADRAT                         <GQ3>
520 PRINT I*I,                          <VAF>
530 RETURN                              <KE7>
```

Listing 2. So können Sie mit Label arbeiten

```
10 REM   *** DEMO 2 ***                     <A99>
20 PRINT CHR$(147)                          <1A3>
30 Z$="ZAHL": Q$="QUADRAT": W$="WURZEL"     <LR3>
40 FOR I = 1 TO 10                          <K0V>
50 GOSUB Z$: GOSUB Q$                       <402>
70 GOSUB W$                                 <1CT>
80 NEXT                                     <GU0>
90 END                                      <RVE>
100 REM ZAHL                                <IIA>
110 PRINT I,                                <3H7>
120 RETURN                                  <4S7>
300 REM WURZEL                              <JIE>
310 PRINT SQR(I)                            <JJV>
320 RETURN                                  <4K7>
510 REM QUADRAT                             <GQ3>
520 PRINT I*I,                              <VAF>
530 RETURN                                  <KE7>
```

Listing 3. Auch hier wurden Zeilensprünge durch Label ersetzt

Tips, Tricks, PEEKs und POKEs zum C 128

Eine wahre Fundgrube für jeden C128-Anwender. Hier finden Sie interessante Listings und Informationen, die Ihnen helfen werden, Ihren C128 besser auszunutzen und kennenzulernen.

Eine wahre Fundgrube für jeden C128-Anwender und -Programmierer haben wir für Sie zusammengestellt. Tips & Tricks-Listings, PEEKs, POKEs und jede Menge Hintergrundinformationen, um Ihnen das Arbeiten mit Ihrem Computersystem zu erleichtern.

Datagenerator C 128

Da der Monitor des C128 im Vergleich mit dem SMON keine Möglichkeit besitzt, Maschinenprogramme in DATA-Zeilen zu verwandeln, mußte der gequälte C128-Besitzer immer zwischen C64- und C128-Modus wechseln, nur um eine kleine Routine in Basic-Zeilen abgelegt zu bekommen.

Mit diesem Datagenerator (Listing 1, bitte mit dem neuen Checksummer 128 eingeben) können bequem alle Maschinensprache-Routinen in Basic-Zeilen übernommen werden.

Das Programm verlangt von Ihnen die Eingabe der Start- und Endadresse des zu wandelnden Bereiches. Diese kann sowohl dezimal oder auch, mit vorangestelltem »$«-Zeichen, hexadezimal erfolgen. Zu beachten ist nur, daß die Hex-Zahl vierstellig sein muß.

Haben Sie die beiden Adressen eingegeben, müssen Sie dem Computer noch die Startzeilennummer der DATA-Zeilen und deren Schrittweite mitteilen. Danach wird der Bildschirm gelöscht und die Einleseschleife übernommen. Wiederum löscht sich der Bildschirm und aus dem angegebenen Bereich werden die Werte in DATA-Zeilen abgelegt, und zwar in hexadezimaler Form. Dadurch sind die erzeugten Zeilen nicht nur schön untereinander formatiert, sondern ersparen den späteren Benutzern auch noch ein Drittel Tipparbeit. Letztendlich löscht sich der Datagenerator per DELETE -40 noch selbst, so daß nun beim LISTen ein speicherfertiges Programm auf dem Bildschirm steht.

```
2 REM ***  DATAGEN 128 ***                     <00P>
4 REM *** (C) T. FAHLE ***                     <00Q>
6 :                                            <01B>
8 SCNCLR                                       <0FJ>
10 INPUT "STARTADR.";SA$: IF MID$(SA$,1,1)="
   $" THEN SA=DEC(RIGHT$(SA$,4)): ELSE SA=VA
   L(SA$)                                      <S5Q>
12 INPUT "ENDADR.";EA$: IF MID$(EA$,1,1)="$"
    THEN EA=DEC(RIGHT$(EA$,4)): ELSE EA=VAL(
   EA$)                                        <JAL>
14 INPUT "1.ZEILENNR.";ZN: INPUT "SCHRITTWEI
   TE";SW: IF ZN<38 THEN 14                    <D09>
16 PRINT CHR$(147);ZN;"FORI="SA"TO"EA"         <0MI>
18 ZN=ZN+SW: PRINT ZN;"READ A$:POKE I,DEC(A$
   )": ZN=ZN+SW: PRINT ZN;"NEXT I"             <G4H>
20 ZN=ZN+SW: PRINT ZN;":"                      <IS6>
22 PRINT "GOTO 26"                             <CMN>
24 POKE 842,19: FOR I=1 TO 5: POKE 842+I,13:
    NEXT : POKE 208,6: END                     <JV1>
26 ZN=ZN+SW: SCNCLR : PRINT ZN;"DATA";         <8MF>
28 FOR I=SA TO SA+14                           <IQT>
30 : PRINT RIGHT$(HEX$(PEEK(I)),2)+",";        <O4D>
32 NEXT I: PRINT CHR$(20)                      <8KP>
34 PRINT "SA="SA":EA="EA":SW="SW":ZN="ZN       <ECK>
36 PRINT "{2DOWN}GOTO38": POKE 842,19: FOR T
   =1 TO 3: POKE 842+T,13: NEXT : POKE 208,5
   : END                                       <RD1>
38 SA=SA+15: IF SA<EA THEN 26                  <UHL>
40 SCNCLR : PRINT "DELETE-40": POKE 842,19:
   POKE 843,13: POKE 208,2: END                <LMA>
```

Listing 1. »DATAGEN 128« – erzeugt ein Basic-Ladeprogramm. Bitte verwenden Sie zur Eingabe den Checksummer 128 auf Seite 122.

```
1000 FORI= 4864 TO 4919
1010 READ A$:POKE I,DEC(A$)
1020 NEXT I
1030 :
1040 DATA78,A9,0E,8D,14,03,A9,13,8D,15,03,58,60,78,A5
1050 DATAD5,F0,25,C5,FC,F0,21,85,FC,A9,15,8D,18,D4,A0
1060 DATA09,A2,00,8C,05,D4,8E,06,D4,A9,30,8D,01,D4,A9
1070 DATA20,8D,04,D4,A9,21,8D,04,D4,58,4C,65,FA,A4,FB
```

Listing 2. So könnte ein mit DATAGEN 128 erzeugtes Listing aussehen

Das Programm »Tasten-Piep« (Listing 2) zeigt, wie ein durch »Datagen 128« bearbeitetes Programm aussieht. Die ersten vier Zeilen zeigen die vom Programm erzeugte Einleseschleife, danach folgen die formatierten DATA-Zeilen. Übrigens können Sie dieses kleine Programm ruhig abtippen, denn einen Piepston auf Tastendruck kann man immer gebrauchen, vor allem dann, wenn man mehr oder weniger blind DATA-Zeilen eintippt.

Wichtiger Hinweis:

Sollten Sie das Programm RENUMBERn, müssen Sie auch in den Zeilen 22, 36 und 40 die Werte entsprechend ändern, da der RENUMBER-Befehl GOTO und DELETE innerhalb einer PRINT-Zeile nicht ändert. (Torsten Fahle/dm)

Punkt-Komma-Tausch

Mit diesem kurzen Programm (Listing 3, bitte mit dem neuen Checksummer 128 eingeben) läßt sich der Dezimalpunkt auf der Zehnertastatur durch ein Komma ersetzen. Dies ist sehr hilfreich, falls lange DATA-Kolonnen mittels der Zehnertastatur eingetippt werden müssen.

Der Dezimalpunkt wird nur im ASCII-Modus mit dem Komma belegt. Es ist aber trotzdem möglich, den deutschen Zeichensatz zu nutzen, wenn man wie folgt vorgeht:

Den C128 ausschalten, auf DIN umstellen, das Programm laden und starten. Nach kurzer Zeit wird man aufgefordert, die Taste F8 zu betätigen.

Nun kann damit begonnen werden, Listings einzugeben. Will man nun den Dezimalpunkt als Komma benutzen, was ja meist in DATA-Zeilen der Fall ist (da hier ja sowieso keine deutschen Umlaute benötigt werden), schaltet man auf ASCII-Zeichensatz um und hat nun das Komma zur Verfügung. Soll wieder Text eingegeben werden, wählt man wieder den DIN-Zeichensatz. Der Dezimalpunkt auf der Zehnertastatur ist nun wieder Dezimalpunkt, kann aber durch erneutes Umschalten auf ASCII wieder als Komma genutzt werden.

Soll nur im ASCII-Modus gearbeitet werden, ist das ganze wesentlich weniger kompliziert. Einfach Programm laden und starten!

Vor dem Speichern eines so eingegebenen Programmes sollte erst einmal <RUN/STOP-RESTORE> betätigt werden, um den Ursprungszustand wieder herzustellen.

Programmbeschreibung

In Zeile 85 bis 105 werden die für das Programm notwendigen Adressen definiert. Die nun folgende FOR..NEXT-Schleife erstellt im RAM ab Adresse $1B00 die später benötigte Normaltabelle des ASCII-Codes, da sie dort einfach zu manipulieren ist. Diese Tabelle ist im Commodore-Handbuch des C128 in Anhang J sehr gut versteckt.

In Zeile 125 wird bei Adresse $0AC5 das siebte Bit gesetzt. Dies bewirkt, daß das High-Byte bei Adresse $033F immer mit dem gesetzten Modus (ASCII oder DIN) übereinstimmt, da die Adresse $033F andernfalls neu gesetzt würde.

Die Zeilen 130 bis 135 setzen nun noch die Adreßzeiger, und zwar $033E auf 0 und $033F auf 27.

Zeile 140 ist eigentlich der komplizierteste Teil. Hier findet der eigentliche Tausch der Tastaturbelegung statt. Sie erhalten nun die Methode gezeigt, mit der sich Tasten nahezu beliebig umbelegen lassen. Es folgt nun eine genaue Erklärung des S. Kohlitz-Tasttausch-Systems:

Man hat durch die FOR..NEXT-Schleife einen ASCII-Zeichensatz im RAM, der bei Adresse 6912 ($1B00) beginnt. Will man nun eine Taste umbelegen, so addiert man zu dem Wert 6912 zuerst den Platz-Code aus der Tastaturtabelle des zu ändernden Zeichens minus 1 (im Falle des Dezimalpunktes also 83-1). Warum denn minus 1? Ganz einfach: Die Platzcodes sind von 1 bis 89 fortlaufend durchnumeriert. Um aber sinnvoll damit arbeiten zu können, sollten sie von 0 bis 88 numeriert sein. Man sagt ja auch acht Bit, die aber von 0 bis 7 durchnumeriert sind.

Wir haben nun den Wert 6912+82 errechnet, also 6994. In diese Speicherstelle POKEt man nun nur noch den Code (nicht den Platzcode) des Zeichens, mit der die Taste belegt werden soll (im Falle des Kommas also eine 44). POKE 6994,44 und der Tausch ist perfekt.

Wollen Sie jetzt noch weitere Tasten umbelegen (wie wäre es mit SHIFT anstatt der Minus- oder Plus-Taste, um die Funktionstasten von der Zehnertastatur leichter zu erreichen?), so dürften der Ausführung keine Probleme mehr im Wege stehen. (Stephan Kohlitz/dm)

Get mit Cursor

Bei allen Commodore-Computern konnte man mit drei kleinen POKEs den Cursor beim GET-Befehl ein- und ausschalten. Beim C128 geht es im Prinzip auch, nur nicht im 80-Zeichen-Modus. Doch nichts ist unmöglich.

Im 80-Zeichen-Modus kann man dem VDC (Videocontroller) ein Zeichen mit einem »Blink-Attribut« versehen übergeben. Wie man anhand des Programms (Listing 4, bitte mit dem neuen Checksummer 128 eingeben) ausprobieren kann, blinkt der Cursor jetzt bei der GET-Routine, egal, ob im 40- oder 80-Zeichen-Modus.

Das Programmsegment für den 80-Zeichen-Modus dürfte durch die reichlichen Kommentare verständlich sein. Beim 40-Zeichen-Modus sind folgende zwei angesprochenen Speicherstellen von Interesse:

2599 $0A27 BLNSW Schalter für den Cursor (204 beim C64); 0 = Ein/1 = Aus

2600 $0A28 BLNCT Zähler für Cursorblinken (205 beim C64)

Den Cursorblinkzähler sollte man nach dem Holen eines Zeichens auf 1 stellen, da sonst hin und wieder ein inverser Block irgendwo auf dem Bildschirm stehenbleibt.

(Michael Bauer/dm)

```
80 RESTORE                                <79M>
85 Z1=DEC("0058"): Z2=DEC("1B00")         <Q9Q>
90 Z3=DEC("0AC5"): Z4=DEC("0080")         <BTV>
95 Z5=DEC("033E"): Z6=DEC("033F")         <UHV>
100 Z7=DEC("001B"): Z8=DEC("1B52")        <CAL>
105 Z9=DEC("002C")                        <N1E>
110 FOR DC=0 TO Z1 : READ DA              <NLK>
115 POKE Z2+DC, DA                        <FN5>
120 NEXT DC                               <5B5>
125 POKE Z3,PEEK(Z3) OR Z4                <UN7>
130 POKE Z5,0                             <KQE>
135 POKE Z6,Z7                            <BV1>
140 POKE Z8,Z9                            <A6U>
145 DATA 20,13,29,136,133,134,135,17      <0F9>
150 :                                     <043>
155 DATA 51,87,65,52,90,83,69,1           <EIU>
160 :                                     <045>
165 DATA 53,82,68,54,67,70,84,88          <1J4>
170 :                                     <047>
175 DATA 55,89,71,56,66,72,85,86          <NQJ>
180 :                                     <04P>
185 DATA 57,73,74,48,77,75,79,78          <17Q>
190 :                                     <04R>
195 DATA 43,80,76,45,46,58,64,44          <OLP>
200 :                                     <045>
205 DATA 92,42,59,19,1,61,94,47           <UIR>
210 :                                     <047>
215 DATA 49,95,4,50,32,2,81,3             <E07>
220 :                                     <041>
225 DATA 132,56,53,9,50,52,55,49          <JVS>
230 :                                     <043>
235 DATA 27,43,45,10,13,54,57,51          <D4P>
240 :                                     <04D>
245 DATA 8,48,46,145,17,157,29,255        <GPP>
250 :                                     <04F>
255 DATA 255                              <ALV>
260 KEY 8,"DELETE-300"+CHR$(13)           <C2U>
270 PRINT CHR$(147);"PUNKT-->KOMMA TAUSCH <NNF>
280 PRINT : PRINT "(C) STEPHAN KOHLITZ 1985" <K4A>
290 PRINT : PRINT "VOR WEITEREN EINGABEN 'F8
    ' DRUCKEN"                            <IDC>
```

Listing 3. Ein Programm zum Austauschen des Punktes mit einem Komma auf der Zehnertastatur

```
100 REM ************************************ <02A>
110 REM *                                  * <02B>
120 REM *    GET MIT BLINKENDEM CURSOR     * <028>
130 REM *    FUER C128 MIT  40 ODER 80     * <029>
140 REM *         ZEICHEN-SCHIRM           * <02E>
150 REM *                                  * <02F>
160 REM ************************************ <02C>
200 PRINT "EINGABE ? ";                    <904>
210 GOSUB 310: IF X$<>"J" AND X$<>"N" THEN 2
    10                                     <M7J>
220 PRINT X$                               <ATI>
230 END                                    <0KU>
250 REM ***************** SUBROUTINE GET   <029>
260 REM ***************** MIT BLINKENDEM   <02A>
270 REM ***************** CURSOR           <02B>
290 REM PRUEFE OB 40 ODER 80 ZEICHEN-SCHIRM <025>
310 IF PEEK(215)=128 THEN BEGIN :   REM 80 Z
    EICHEN                                 <IBH>
330 : REM CHR$(15)  = ZEICHENBLINK AN      <047>
340 : REM CHR$(18)  = RVS ON               <049>
350 : REM CHR$(146) = RVS OFF              <04B>
360 : REM CHR$(157) = CURSOR LINKS         <04D>
370 : REM CHR$(143) = ZEICHENBLINK AUS     <04F>
390 :  PRINT CHR$(15) CHR$(18);" ";CHR$(146)
     CHR$(157) CHR$(143);                  <DUV>
410 :  GET KEY X$:     REM WARTE AUF TASTEND
    RUCK                                   <09K>
430 BEND : ELSE BEGIN : REM 40 ZEICHEN     <V4R>
450 :  POKE 2599,0: REM SCHALTE BLINKEN EIN <PSD>
470 :  GET KEY X$:  REM WARTE AUF TASTENDRUC
    K                                      <69K>
```

Listing 4. Nun blinkt der Cursor auch bei GET. Bitte beachten Sie den Checksummer 128 auf Seite 122.

Bildschirme verwalten

Dieses Programm macht es möglich, bis zu drei Bildschirme gleichzeitig im 80-Zeichen-Modus zu verwalten. Die einzelnen Bildschirme können nach dem Starten des Basic-Laders mit SYS 2883, Sichtbar, Bildschirmnummer, Attribut aufgerufen werden. Dabei haben die Parameter folgende Bedeutung:

– Sichtbar: Sichtbar kann zwei verschiedene Werte annehmen.

Wert 0: Der vorherige Bildschirm wird angezeigt. Die Befehle PRINT und CHAR schreiben aber schon auf den neuen Bildschirm. Dadurch kann ein Bildschirm aufgebaut werden, während ein anderer angezeigt wird.

Wert 1: Der angewählte Bildschirm wird auch tatsächlich angezeigt.

– Bildschirm: Es stehen die Bildschirme 1 bis 5 zur Verfügung. Bitte lassen Sie sich nicht verwirren, die Bildschirme 1 und 3 sind die gleichen.

– Attribut: Auch für das Attribut gibt es zwei Werte:

Wert 0: Attribut wird ausgeschaltet. Das heißt, auf dem Bildschirm sind keine Farben, blinkende und unterstrichene Zeichen mehr sichtbar. Außerdem werden alle Zeichen im Grafikmodus angezeigt. Das alles gilt auch, wenn Sie den Bildschirm nicht sichtbar einschalten. Beim Beschreiben des Bildschirms wird aber auch das abgeschaltete Attribut-RAM beschrieben. Diese Funktion (Attribut aus) ist für die Bildschirme 3, 4 und 5 gedacht, für die nur ein Attribut-RAM zur Verfügung steht.

Wert 1: Attribut wird eingeschaltet.

Insgesamt können fünf Bildschirme erzeugt werden. Die verschiedenen Bildschirme werden durch Verschiebung des Video-RAMs (in ihm stehen die Bildschirmcodes der Zei-

Name	Nummer	Adresse	Funktion
FSTMODE	01	$ff47	Fastimpuls an Bus
EAINIT	02	$ff4a	Standard E/A-Geräte setzen
C64MODE	03	$ff4d	Aktivierung des 64'er-Modus
DMA-CALL	04	$ff50	Externe RAM-Bausteine initialisieren
BOOT-CALL	05	$ff53	Booten der Diskette
PHOENIX	06	$ff56	Kaltstart
LKUPLA	07	$ff59	Suche nach logischer Filenummer
LKUPSA	08	$ff5c	Suche nach Sekundäradresse
SWAPPER	09	$ff5f	Umschalter zwischen 40 und 80 Zeichen
DLCHR	10	$ff62	Kopiert Zeichensatz ins VDC-RAM
PFKEY	11	$ff65	F-Tastenbelegung
SETBANK	12	$ff68	Speicherbank für LOAD/SAVE definieren
GETCONF	13	$ff6b	Holt Konfigurationsbyte
JSRFAR	14	$ff6e	JSR in beliebige Bank
JMPFAR	15	$ff71	JMP in beliebige Bank
FETCH	16	$ff74	LDA ( ),Y aus beliebiger Bank
STASH	17	$ff77	STA ( ),Y in beliebiger Bank
COMPARE	18	$ff7a	CMP ( ),Y in beliebiger Bank
PRIMM	19	$ff7d	Textausgabe
CINT	20	$ff81	Editor-Initialisierung
IOINIT	21	$ff84	Ein-/Ausgabe-Initialiesierung
RAMTAS	22	$ff87	Warmstart
RESTOR	23	$ff8a	Systemvektoren initialisieren
VECTOR	24	$ff8d	Systemvektoren kopieren
SETMSG	25	$ff90	Systemmeldungen ermöglichen/ verhindern
SECND	26	$ff93	Sekundäradresse nach LISTn
TKSA	27	$ff96	Sekundäradresse nach TALK
MEMTOP	28	$ff99	Lesen/setzen der Speicherobergrenze
MEMBOT	29	$ff9c	Lesen/setzen der Speicheruntergrenze
KEY	30	$ff9f	Tastaturabfrage
SETTMO	31	$ffa2	Setzen des Timeoutflags für IEEE
ACPTR	32	$ffa5	Holt Byte vom Bus
CIOUT	33	$ffa8	Legt Byte auf Bus
UNTLK	34	$ffab	Untalk an Bus senden
UNLSN	35	$ffae	Unlisten an Bus senden
LISTN	36	$ffb1	Listen an Bus senden
TALK	37	$ffb4	Talk an Bus senden
READST	38	$ffb7	Statusbyte lesen
SETLFS	39	$ffba	Setzen der Fileparameter
SETNAM	40	$ffbd	Setzen des Filenamens
OPEN	41	$ffc0	Datei öffnen
CLOSE	42	$ffc3	Datei schließen
CHKIN	43	$ffc6	Datei ist Eingabedatei
CKOUT	44	$ffc9	Datei ist Ausgabedatei
CLRCH	45	$ffcc	Ein-/Ausgabekanäle schließen
BASIN	46	$ffcf	Zeicheneingabe
BSOUT	47	$ffd2	Zeichenausgabe
LOAD	48	$ffd5	Laden einer Datei
SAVE	49	$ffd8	Speichern einer Datei
SETTIM	50	$ffdb	Setzen der internen Uhr
RDTIM	51	$ffde	Lesen der internen Uhr
STOP	52	$ffe1	Stop-Tasten-Abfrage
GETIN	53	$ffe4	Holt Zeichen aus Tastaturpuffer
CLALL	54	$ffe7	Schließen aller Dateien
UDTIM	55	$ffea	Interne Uhr hochzählen
SCRORG	56	$ffed	Fenstergröße feststellen
PLOT	57	$fff0	Cursorposition holen/setzen
IOBASE	58	$fff3	Holt die Basisadresse des I/O-Bereiches

Tabelle 1. Die wichtigsten Kernel-Routinen und ihre Funktion

```
10 : REM BILDSCHIRME                                <02F>
20 :                                                <029>
30 : REM MASCHINENSPRACHEDATEN EINLESEN             <02B>
40 : BANK 0: FAST                                   <RQI>
50 : FOR I = DEC("B00") TO DEC ("B91")              <58T>
60 :  READ DA$: POKE I, DEC(DA$)                    <IMQ>
70 :  SU = SU + DEC(DA$)                            <LV5>
80 : NEXT I                                         <204>
90 : IF SU <> 11931 THEN PRINT "{CTRL+O}FEHL
   ER IN DATAZEILEN": END : ELSE BEGIN              <4UV>
100 :  PRINT "{CTRL+O}DATAZEILEN SIND RICHTI
    G.": SLEEP  : BEND                              <V8P>
110 : REM DIE STEUERZEICHEN IN ZEILE70/80 SI
    ND < CONTROL O >                                <04B>
120 BANK 15: SYS DEC("C07B"): REM BILDSCHIRM
     INITIALISIEREN                                 <CSF>
130 :                                               <04F>
140 : REM ASSEMBLERDATEN                            <041>
150 DATA 8E,00,D6,2C,00,D6,10,FB,8D,01,D6,60
    ,8E,00,D6,2C                                    <KL6>
160 DATA 00,D6,10,FB,AD,01,D6,60,A9,00,E0,01
    ,D0,07,A2,0C                                    <5IG>
170 DATA 20,00,0B,A2,01,8D,2E,0A,60,A9,08,E0
    ,01,D0,05,A2                                    <LOK>
180 DATA 14,20,00,0B,8D,2F,0A,60,20,0C,0B,09
    ,40,20,00,0B                                    <57F>
190 DATA 4C,53,0B,48,8A,48,A2,19,98,D0,ED,20
    ,0C,0B,29,BF                                    <MOG>
200 DATA 20,00,0B,68,C9,01,F0,04,C9,03,D0,08
    ,68,AA,20,18                                    <ASI>
210 DATA 0B,4C,29,0B,C9,02,D0,0C,68,AA,A9,10
    ,20,1A,0B,A9                                    <Q2L>
220 DATA 18,4C,2B,0B,C9,04,D0,0A,68,AA,A9,10
    ,20,1A,0B,4C                                    <DU2>
230 DATA 29,0B,C9,05,D0,0A,68,AA,A9,18,20,1A
    ,0B,4C,29,0B                                    <81Q>
240 DATA 68,60                                      <BFM>
250 :                                               <04F>
260 : REM MIT SYS2883,SICHTBAR,BILDSCHIRM,AT
    TRIBUT WERDEN DIE BILDSCHIRME                   <049>
270 : REM EINGESCHALTET. SICHTBAR KANN DIE W
    ERTE 0 (NICHT SICHTBAR) UND 1 (SICHT-           <04B>
280 : REM BAR) ANNEHMEN. ICHT SICHTBAR HEISS
    T, DASS FUER DIE BEFEHLE PRINT UND              <04L>
290 : REM CHAR AUF DEN NEUEN BILDSCH. ZUGEGR
    IFFEN WIRD, ABER GLEICHZEIT. DER ALTE           <04N>
300 : REM BILDSCH. ANGEZEIGT WIRD. BILDSCHIR
    M KANN DIE WERTE VON 1-5 ANNEHMEN.              <041>
310 : REM ATTRIBUT KANN DIE WERTE 0 (AUS) UN
    D 1 (EIN) ANNEHMEN.                             <043>
320 : REM DER BEFEHL IN ZEILE 120 ZUM INITIA
    LISIEREN DES BILDSCHIRMS IST NOTWEN-            <045>
330 : REM DIG UM DAS PRG. BILDSCHIRME FEHLER
    FREI ABLAUFEN ZU LASSEN. ER MUSS EIN-           <047>
340 : REM MAL NACH DEM POKEN DER MASCHINENSP
    RACHEDATEN AUSGEFUEHRT WERDEN.                  <049>
```

Listing 5. Mehrere Bildschirme austauschen

chen, die auf dem Bildschirm sichtbar sind), und durch Verschieben beziehungsweise Abschalten des Attribut-RAMs (in ihm werden die Farben und alternativen Darstellungen eines Zeichens eingetragen) des VDC-RAM (der VDC besitzt ein eigenes, 16 KByte großes RAM) realisiert.

Dabei stehen folgende Bildschirme zur Verfügung:

Bildschirm 1	Video-RAM von $0000 bis $0800, Attribut-RAM von $0800 bis $1600
Bildschirm 2	Video-RAM von $1600 bis $2400, Attribut-RAM von $2400 bis $3200
Bildschirm 3	Video-RAM von $0000 bis $0800, Attribut-RAM von $0800 bis $1600
Bildschirm 4	Video-RAM von $1600 bis $2400, Attribut-RAM von $0800 bis $1600
Bildschirm 5	Video-RAM von $2400 bis $3200, Attribut-RAM von $0800 bis $1600

Daraus ergibt sich die folgende Aufteilung: Bildschirm 1 und 2 oder Bildschirm 3, 4 und 5 können zusammen benutzt werden, ohne daß Verluste des Bildschirminhalts auftreten. Bei den Bildschirmen 3, 4 und 5 wird dasselbe Attribut-RAM benutzt. Für eben diese Schirme steht dann jedoch nur ein Attribut-RAM zur Verfügung.

Eingabehinweise

Das Programm (Listing 5) geben Sie bitte mit dem neuen Checksummer 128 ein und speichern es. Das Programm starten Sie durch RUN. Das zweite Programm (Listing 6) ist ein Demo-Programm, das nicht unbedingt abgetippt werden muß.

Der Befehl SYS DEC ("C07B") im Basic-Lader ist nötig, um den Bildschirm zu initialisieren. Wird dieser Befehl nicht ausgeführt, so kann es geschehen, daß sich der Computer nach dem Wechsel eines Bildschirms nicht mehr meldet.

```
10 FAST : EIN=1: AUS=0                             <D98>
11 :                                               <02E>
20 BS=2: SYS 2883,EIN,BS,EIN: PRINT "{CLR}":
            REM BILD 2 LOESCHEN U EINSCHALTEN      <5K1>
30 BS=1: SYS 2883,AUS,BS,EIN: PRINT "{CLR}":
      REM BILD 1 LOESCHEN U NICHT SICHTB. EIN      <9LM>
31 :                                               <02A>
40 FOR I=1 TO 10                                   <JSV>
50 PRINT "{CTRL+B}CHRISTOPH FRANZEN BONIFATI
   USSTRASSE 70 4130 MOERS ASBERG BILDSCHIRM
    1                                              <SCS>
60 NEXT                                            <0P0>
61 :                                               <020>
70 BS=1: SYS 2883,EIN,BS,EIN:
                 REM BILD I EINSCHALTEN            <VAA>
80 INPUT "BITTE GEBE DEINEN NAMEN EIN"; NA$        <TLH>
81 PRINT "{CLR,12SPACE}HALLO "NA$"{2DOWN}          <195>
82 PRINT "ICH HOFFE ES GEHT DIR HEUTE SEHR G
   UT. JEDENFALLS WIRD JETZT GERADE WAEHREND       <K8F>
83 PRINT "DU DIESEN TEXT LIESST DEIN NAME AU
   F BILDSCHIRM 2 BLINKEND GESCHRIEBEN.            <K07>
84 PRINT "WENN DU GLEICH DIE AUFFORDERUNG BE
   KOMMST EINE TASTE ZU DRUECKEN, DANN IST         <INK>
85 PRINT "DER AUFBAU DES BILDSCHIRMS 2 FERTI
   G UND DU KANNST IHN DIR ANSEHEN, INDEM DU       <T1D>
86 PRINT "DER AUFFORDERUNG NACHKOMMST UND EI
   NE BELIEBIGE TASTE DEINES COMPUTERS             <S7E>
87 PRINT "DRUECKST.                                <8DS>
89 :                                               <02K>
90 BS=2: SYS 2883,AUS,BS,EIN  :
         REM BILD 2 NICHT SICHTB. EIN              <QEI>
100 PRINT "{CLR,CTRL+B}CHRISTOPH FRANZEN BON
    IFATIUSSTRASSE 70 4130 MOERS ASBERG BILD
    SCH.2                                          <US5>
110 FOR I=0 TO 100: PRINT " {CTRL+O}" NA$ "
    "; : NEXT                                      <BAR>
111 :                                              <04A>
120 BS=1: SYS 2883,EIN,BS,EIN:
          REM BILD 1 WIEDER EINSCHALTEN            <6IA>
130 CHAR 1,20,20, "BITTE EINE TASTE DRUECKEN
    "                                              <IQ2>
140 GET KEY A$                                     <KCD>
141 :                                              <040>
150 BS=2: SYS 2883,EIN,BS,EIN:
       REM BILD 2 ZUM SEHEN EINSCHALTEN            <7AI>
```

Listing 6. Ein Demo-Programm, das Ihnen die Fähigkeiten des Bildschirm-Vertauschens aufzeigt

Dann müssen Sie das Programm mit RUN/STOP-RESTORE unterbrechen. (Christoph Franzen/dm)

Programme verbinden (MERGE)

Mit diesem Programm (Listing 7, bitte mit dem neuen Checksummer 128 eingeben) lassen sich mehrere Programme zu einem einzigen verbinden.

Das Programm »Merge« ist so geschrieben, daß es an einen freien Speicherplatz gesetzt und aufgerufen werden kann. Um das Programm an eine andere Adresse zu verschieben, muß im Basic-Lader in Zeile 30 die Variable »AN« geändert werden.

```
10 REM MERGE                                       <019>
20 :                                               <029>
30 AN = 2816 : BANK 0                              <3NT>
40 :                                               <025>
50 REM MASCHINENSPRACHEDATEN EINLESEN              <01D>
60 FOR I = AN TO AN + 30                           <G9H>
70 READ DA$: POKE I, DEC(DA$)                      <KAQ>
80 SU = SU + DEC(DA$)                              <5VT>
90 NEXT                                            <0UG>
100 IF SU <> 2733 THEN PRINT "FEHLER IN DATA
    ZEILEN": END : ELSE BEGIN                      <L78>
110 PRINT "PROGRAMMSTART AN PROGRAMMENDE SET
    ZEN MIT SYS" AN                                <7S3>
120 PRINT "PROGRAMME ZU EINEM PROGRAMM ZUSAM
    MENSETZEN MIT SYS" AN+22                       <UE1>
130 :                                              <04F>
140 REM MASCHINENSPRACHEDATEN                      <02E>
150 DATA AD,10,12,38,E9,02,85,2D,AD,11,12,E9
    ,00,85,2E,A0                                   <PSH>
160 DATA 02,A9,00,91,2D,60,A9,01,85,2D,A9,1C
    ,85,2E,60                                      <9LS>
170 :                                              <047>
180 REM ENDE                                       <022>
```

Listing 7. »Merge« – Verbinden mehrerer Basic-Programme

Die Bedienung der Routine:

1) Reset-Taster am Computer betätigen beziehungsweise Computer aus- und einschalten. Das ist wichtig, um sicherzugehen, daß der Basic-Start wirklich bei $1C00 beginnt.

2) Programm MERGE laden und starten. Bitte schreiben Sie sich die Adressen der SYS-Befehle auf.

3) Laden Sie Ihr Programm, das Sie mit einem anderen verbinden wollen, und bringen Sie das Programm eventuell durch RENUMBER in einen vernünftigen Zeilennummernbereich. Schreiben Sie sich bitte die höchste Zeilennummer des ersten Programmes auf.

4) Verschieben Sie den Basic-Start (SYS 2816).

5) Laden Sie das nächste Programm und bringen auch dieses durch RENUMBER in einen vernünftigen Zeilennummernbereich (die Zeilennummern des zweiten Programms müssen höher als die des ersten sein).

6) Verbinden Sie die Programme (SYS 2838).

7) Nun können Sie das neue Programm speichern.

Nach dem Verschieben des Basic-Starts sind alle Befehle erlaubt. Das heißt, Sie können die geladenen Programme mittels DELETE, RENUMBER oder sonstigen Befehlen verändern. (Christoph Franzen/dm)

Bankswitching in Assembler

Durch das Bankswitching wird die Assemblerprogrammierung oft sehr aufwendig und mühselig. Wer schon einmal versucht hat, aus BANK 1 in eine Kernel-Routine in BANK 15 zu springen, wunderte sich sicher, denn danach befand sich der Computer in einem (sehr) undefinierten Zustand.

Um diesem Problem zu begegnen, müßte ein Programm her, das vor Aufruf die gewünschte Konfiguration einschaltet und bei der Rückkehr die alte Speicherbelegung wiederherstellt.

Das Programm müßte zusätzlich in der Common-Area (jenem Speicherbereich, der für alle BANKs gleich ist) funktionieren. Es bietet sich der RS 232-Eingabepuffer von $C00 bis $D00 an.

Das Programm »Kernel« (Listing 8), das in diesem Bereich liegt, funktioniert so:

1) Die Nummer der Kernel-Routine wird in 996 gespeichert (die Nummern entnehmen Sie bitte Tabelle 1)

2) Akku, X- und Y-Register so mit ihren Werten laden, als würde direkt in die gewünschte Kernel-Routine verzweigt

3) JSR $C00

Manchmal kommt es auch vor, daß ein Unterprogramm im Basic-ROM von $F4000 bis $FAFFF oder im Monitor von $FB000 bis FBFFF aufgerufen werden soll. Solch ein Problem ist mit JSRFAR zu lösen:

1) Die anzuspringende BANK ist in $02 abzulegen
2) Die anzuspringende Adresse ist im High-/Low-Format in $03/$04 bereitzustellen
3) Akku in $06, X-Register in $07 und Y-Register in $08
4) JSR $C23

Soll kein Unterprogrammaufruf, sondern ein normaler Sprung stattfinden, so ist Punkt 4 durch JMP $CD0 zu ersetzen.

Diese seltsame Art der Parameterübergabe entspricht den original JMP- und JSRFAR-Routinen. Es dürfte also keine großen Probleme bei der Umrüstung bereits bestehender Programme geben.

»Kernel 128« muß vor dem Start noch initialisiert werden. Dies erfolgt durch JSR $C3E. Findet der Interpreter während der Ausführung eines Basic-Programms eine USR(X)-Anweisung, so wird nach $0F000 gesprungen.

(Frank Probst/dm)

Wichtiger Hinweis zum Abtippen
Bitte beachten Sie den neuen Checksummer 128. Mit diesem Programm können Sie alle C128-Basic-Listings eingeben. Die Prüfsumme am Ende jeder Basic-Zeile unterrichtet Sie sicher, ob Sie die Zeile richtig eingegeben haben. Den Checksummer 128 finden Sie in dieser Ausgabe.

```
name : kernel 128        0c00 0d10
-----------------------------------
0c00 : 85 06 86 07 84 08 a9 0f  58
0c08 : 85 02 a9 44 85 04 a9 ff  a0
0c10 : 85 03 ad e4 03 c9 14 90  0f
0c18 : 02 e6 04 0a 6d e4 03 65  a5
0c20 : 04 85 04 ad 00 ff 48 a9  12
0c28 : 00 8d 00 ff 20 cd 02 68  38
0c30 : 8d 00 ff a5 05 48 a5 06  a7
0c38 : a6 07 a4 08 28 60 a9 00  b8
0c40 : 8d 00 ff ad 06 d5 09 05  c0
0c48 : 8d 06 d5 a9 00 85 2f 85  77
0c50 : 31 a9 10 85 30 85 32 a9  56
0c58 : 72 8d 19 12 a9 0c 8d 1a  7f
0c60 : 12 a9 7f 8d 00 ff a0 37  c9
0c68 : b9 d8 0c 99 50 ff 88 10  0b
0c70 : f7 60 a9 3f 8d 00 ff 4c  5b
0c78 : 00 f0 8d 8d 0c 8d 92 0c  95
0c80 : ee 92 0c ad 00 ff 48 a9  e5
0c88 : 7e 8d 00 ff a5 00 8d 5a  12
0c90 : ff a5 00 8d 5b ff 20 50  ea
0c98 : ff 8d e4 03 68 8d 00 ff  ea
0ca0 : ad e4 03 60 8d e4 03 a9  ec
0ca8 : ff 8d bc 0c 8d c1 0c ee  13
0cb0 : c1 0c ad 00 ff 48 a9 7e  c9
0cb8 : 8d 00 ff a5 00 8d 7c ff  58
0cc0 : a5 00 8d 7d ff ad e4 03  7f
0cc8 : 20 6c ff 68 8d 00 ff 60  c5
0cd0 : a9 00 8d 00 ff 4c e3 02  d3
0cd8 : 78 ad 06 d5 29 f3 8d 06  d8
0ce0 : d5 b9 ff ff 8d f0 ff ad  4d
0ce8 : 06 d5 09 04 8d 06 d5 ad  57
0cf0 : f0 ff 58 60 8d f0 ff 78  53
0cf8 : ad 06 d5 29 f3 8d 06 d5  b2
0d00 : ad f0 ff 99 ff ff ad 06  1b
0d08 : d5 09 04 8d 06 d5 58 60  46
```

Listing 8. »Kernel 128« – Bitte mit dem MSE im C64-Modus eingeben

Boot-Sektoren selbst erstellen

Die Wirkung eines Boot-Sektors kennt wohl jeder Besitzer eines C128. Doch eine Diskette mit diesem Boot-Sektor (Listing 9) versehen, ist dagegen ganz etwas anderes.

Selbst mit dem »Auto-Boot-Creater« von der Test-Demo-Diskette kann leider nur ein Boot-Sektor mit Autostart für ein Maschinen- oder ein Basic-Programm erzeugt werden.

Oder man studiert die entsprechenden Kapitel in den entsprechenden Fachbüchern, und schreibt den Boot-Sektor mit Floppy-Befehlen oder einem Disketten-Monitor (Wer hat den schon für den C128?) auf die Diskette. Aber versuchen Sie dann einmal, das so geladene Basic-Programm im Speicher zu verändern! Oder lieber nicht, denn absolut geladene Basic-Programme lassen sich nicht mehr editieren. Doch das steht ja schon im »Commodore-128-Handbuch« von Markt & Technik.

Doch Gott sei Dank gibt es noch vernünftige Boot-Maker wie den hier vorgestellten.

Hiermit lassen sich leicht und locker 17 (in Worten: siebzehn) verschiedene Boot-Kombinationen erstellen!

Zunächst einmal kann man frei wählbaren Text während des Ladens auf den Bildschirm geben. Hierbei sind sogar Steuerzeichen möglich. Oder man gibt schlicht den Namen des Programms ein, das gerade geladen wird.

Das zu ladende Programm kann mit oder ohne Autostart sein. Natürlich kann man auch nur einen Begrüßungstext auf den Bildschirm ausgeben. Oder nur ein Maschinen- oder Basic-Programm laden.

Aus den drei Grundeinstellungen

1) Text	nein / ja
2) Masch.Prg.	nein / ja, mit Autostart
	ja, ohne Autostart
3) Basic-Prg.	nein / ja, mit Autostart
	ja, ohne Autostart

ergeben sich die oben genannten 17 Boot-Versionen.

»Boot-Sektor« wurde auf einem 80-Zeichen-Montior erstellt, aber auch auf einem 40-Zeichen-Monitor bleibt die Übersichtlichkeit erhalten. Hierfür wird vom Programm automatisch festgestellt, welcher der Bildschirme gerade eingeschaltet ist.

Selbstverständlich läuft das Programm nicht nur mit den Floppy-Laufwerken 1570/1571, sondern auch mit der guten, alten 1541.

Zum Ausprobieren nimmt man am besten eine frisch formatierte Diskette mit je einem Testprogramm in Basic und Maschinensprache. Daß die Floppy 1571 beidseitig formatierte Disketten schneller liest, hat sich sicher schon herumgesprochen; es entfallen dann nämlich einige vergebliche Leseversuche. Aber das nur nebenbei.

Doch nun zur eigentlichen Beschreibung des Programms (Listing 9), das Sie übrigens mit dem neuen Checksummer 128 eingeben müssen:

Zeile 100 löscht den Bildschirm und aktiviert die Fehlerfalle. Hiermit soll vor allem ein Programmabbruch durch versehentliches Drücken der <STOP>-Taste oder falsche HEX-Eingabe (in Zeile 530) vermieden werden.

Es folgen in den Zeilen 120 bis 150 Initialisierung und Test auf Schreibschutz.

Mit Zeile 160 bis 200 wird der Sektor 0 in Spur 1 anhand der ersten 3 Byte geprüft. Je nach Ergebnis kann nach einer Information laut Zeile 210 oder Zeile 250 das Programm in Zeile 290/320 abgebrochen oder fortgesetzt werden. Hier kommen übrigens die sehr praktischen, für C64-Aufsteiger neuen Basic-Anweisungen GETKEY und IF..THEN..ELSE zur Anwendung.

Haben die ersten 3 Byte den Wert 00, ist der Block frei und es wird von Zeile 190 direkt nach Zeile 340 gesprungen.

```
10 REM : : : : : : : : : : : : : : : : : :
    : : : : : : : : : : : : : :                 <02F>
20 REM  PROGRAMM ZUR EINRICHTUNG EINES          <01A>
30 REM      BOOT-SEKTORS  FUER DEN              <01B>
40 REM         COMMODORE 128 PC                 <01C>
90 REM : : : : : : : : : : : : : : : : : :
    : : : : : : : : : : : : : :                 <02V>
100 PRINT "{CLR}": TRAP 1080                    <Q7I>
110 PRINT "{3SPACE}BOOT-SEKTOR FUER DEN COMM
    ODORE 128{9SPACE}(C) 1986{2SPACE}HANS-GE
    ORG SCHMIDT"                                <D5J>
120 OPEN 15,8,15                                <ES1>
130 PRINT#15,"IO": PRINT#15,"B-F 0 1 0"         <F1E>
140 IF VAL(DS$)<>0 THEN 1040                    <UNL>
150 PRINT#15,"B-A 0 1 0": OPEN 5,8,5,"#"        <KVD>
160 PRINT#15,"U1 5 0 1 0"                       <KIN>
170 PRINT#15,"B-P 5 0"                          <MU7>
180 GET #5,A$,B$,C$: DCLOSE                     <JTR>
190 IF A$+B$="" THEN 340                        <KB3>
200 IF A$+B$+C$="CBM" THEN 250                  <JK6>
210 PRINT "{3DOWN,2SPACE}ACHTUNG!"              <HJ4>
220 PRINT "{2DOWN,2SPACE}DER BOOTSEKTOR{2SPA
    CE}(SEKTOR 0{2SPACE}IN SPUR 1){3SPACE}IS
    T NICHT FREI !"                             <KD9>
230 PRINT "{2DOWN,2SPACE}WICHTIGE DATEN ZUM
    RETTEN ?"                                   <J2L>
240 PRINT "{DOWN,2SPACE}ODER WEITERMACHEN?":
     GOTO 290                                   <37M>
250 PRINT "{3DOWN} ACHTUNG!"                    <KTB>
260 PRINT "{DOWN} AUF DER DISKETTE BEFINDET
    SICH BEREITS{2SPACE}EIN BOOT-SEKTOR!"       <H14>
270 PRINT "{DOWN} UEBERSCHREIBEN ? ?"           <HDB>
280 PRINT "{DOWN} ODER PROGRAMM ABBRECHEN ?"    <PSD>
290 PRINT "{2DOWN,2SPACE}X =ABBRECHEN{9SPACE
    }W = WEITER"                                <N7B>
300 GET KEY WX$                                 <8QO>
310 IF WX$="W" THEN 340                         <LG8>
320 IF WX$="X" THEN 1120: ELSE 300              <3K1>
330 :                                           <047>
340 CLR : PRINT "{2HOME,3DOWN}";CHR$(27);"@{
    2DOWN}"                                     <FOE>
350 PRINT "{2SPACE}KEIN TEXT = NUR {RVSON}RE
    TURN{RVOFF} !{3UP}"                         <KAA>
360 POKE 842,34: POKE 208,1                     <R6P>
370 INPUT "{2SPACE}TEXT HINTER 'BOOTING'{15S
    PACE}";TX$                                  <1QE>
380 IF TX$="" THEN PRINT TAB(INT(PEEK(215)/3
    .2))"{UP}...."                              <CET>
390 :                                           <04J>
400 INPUT "{DOWN,2SPACE}NAME DES ZU LADENEN
    MASCH-PROGRAMMS ";MP$                       <1DE>
410 IF MP$="" THEN RU=LEN(TX$)+19: PRINT "{3
    UP}": GOTO 650                              <0NJ>
420 PRINT "{DOWN,2SPACE}SOLL DAS MASCHINEN-P
    ROGRAMM"                                    <8IG>
430 PRINT "{2SPACE}AUTOMATISCH GESTARTET WER
    DEN{2SPACE}?{2SPACE}J/N"                    <S50>
440 GET KEY  AM$                                <DR1>
450 IF AM$="J" OR AM$="N" THEN 460: ELSE 440    <SS8>
460 IF AM$="J" THEN JN=22: ELSE JN=19           <GSF>
470 RU=LEN(TX$)+LEN(MP$)+JN                     <TCQ>
480 IF AM$="J" THEN 500                         <446>
490 PRINT TAB(PEEK(215)/3.2+17)"{4UP}= KEIN
    AUTOSTART";CHR$(27) CHR$(64): GOTO 650      <8HV>
500 PRINT TAB(PEEK(215)/3.2+17)"{4UP}= MIT A
    UTOSTART" CHR$(27) CHR$(64)                 <A88>
510 PRINT "{DOWN}": TRAP 1080                   <R71>
520 :                                           <04T>
530 INPUT "{UP,2SPACE}ADRESSE IN HEX{5SPACE}
    N{3LEFT}";HE$                               <TI9>
540 IF HE$="N" THEN 580                         <GK0>
550 SY=DEC(HE$)                                 <H68>
560 PRINT TAB(26)"{UP}= SYS";SY                 <L7I>
570 GOTO 620                                    <BAA>
580 INPUT "{UP,2SPACE}ADRESSE IN DEZIMAL (SY
    S)";SY                                      <1PP>
590 IF SY=0 THEN PRINT "{UP}" CHR$(27)"D": G
    OTO 530                                     <VUK>
600 IF SY>65535 THEN PRINT TAB(28)"{UP}=ZU G
    ROSS!": ELSE 620                            <QM8>
610 SLEEP 2: PRINT "{UP}" CHR$(27)"D": GOTO
    580                                         <V3J>
620 JH=INT(SY/256)                              <PGV>
630 JL=SY-JH*256                                <GB7>
640 :                                           <04T>
650 INPUT "{DOWN,2SPACE}NAME DES ZU LADENEN
    BASIC-PROGRAMMS ";BP$                       <1E3>
660 IF BP$="" THEN PRINT "{UP}";CHR$(27)"D":
    GOTO 750                                    <GF8>
670 PRINT "{DOWN,2SPACE}SOLL DAS BASIC-PROGR
    AMM"                                        <50G>
680 PRINT "{2SPACE}AUTOMATISCH GESTARTET WER
    DEN{2SPACE}?{2SPACE}J/N"                    <451>
690 GET KEY  AB$                                <9BE>
700 IF AB$="J" OR AB$="N" THEN 710: ELSE 690 <0NV>
710 IF AB$="J" THEN BS$="RUN": ELSE BS$="DLO
    AD"                                         <99D>
720 IF AB$="J" THEN 740                         <Q4A>
730 PRINT TAB(PEEK(215)/3.2+17)"{4UP}= KEIN
    AUTOSTART";CHR$(27) CHR$(64): GOTO 750      <EHT>
740 PRINT TAB(PEEK(215)/3.2+17)"{4UP}= MIT A
    UTOSTART" CHR$(27) CHR$(64)                 <A88>
750 IF TX$="" AND MP$="" AND BP$="" THEN 112
    0                                           <V7O>
760 :                                           <04T>
770 PRINT "{3DOWN} ALLES OK ?";CHR$(27)"Q"      <S98>
780 PRINT "{DOWN} DIE RICHTIGE DISKETTE IM L
    AUFWERK ? ?"                                <T6D>
790 PRINT "{DOWN,4SPACE}J = WEITER{8SPACE}N
    = NEUBEGINN{9SPACE}X = ABBRUCH"             <CA3>
800 GET KEY  OK$                                <3RR>
810 IF OK$="J" OR OK$="N" OR OK$="X" THEN 82
    0: ELSE 800                                 <N7L>
820 IF OK$="N" THEN 340                         <GKS>
830 IF OK$="X" THEN 1120                        <4P3>
840 :                                           <055>
850 CB$="CBM"                                   <34K>
860 NB$=CHR$(0)                                 <FEM>
870 :                                           <053>
880 OPEN 15,8,15: OPEN 5,8,5,"#"                <DUE>
890 PRINT#15,"U1 5 0 1 0": PRINT#15,"B-P 5 0
    "                                           <L9P>
900 PRINT#5,CB$;NB$;NB$;NB$;NB$;                <V51>
910 PRINT#5,TX$;NB$;MP$;NB$;                    <H2K>
920 IF AM$="N" OR MP$="" THEN 940               <JEU>
930 PRINT#5,CHR$(32);CHR$(JL);CHR$(JH);         <SAC>
940 IF BP$="" THEN BP$=CHR$(96): GOTO 990       <KPI>
950 PRINT#5,CHR$(169);CHR$(RU);CHR$(133);CHR
    $(61);                                      <18A>
960 PRINT#5,CHR$(169);CHR$(11);CHR$(133);CHR
    $(62);                                      <JPC>
970 PRINT#5,CHR$(76);CHR$(209);CHR$(77);        <4E3>
980 PRINT#5,BS$;CHR$(34);                       <F11>
990 PRINT#5,BP$;CHR$(0)                         <R00>
1000 PRINT#15,"U2 5 0 1 0"                      <IAN>
1010 CLOSE 5: CLOSE 15: TRAP                    <0J7>
1020 PRINT "{5UP,2SPACE}" CHR$(27)"@"DS$        <V5S>
1030 :                                          <083>
1040 IF VAL(DS$)=26 THEN PRINT "{2DOWN,2SPAC
     E}BITTE SCHREIBSCHUTZ ENTFERNEN": ELSE
     1060                                       <F7B>
1050 PRINT "{DOWN,2SPACE}UND NEU STARTEN !":
      END                                       <IMD>
1060 IF VAL(DS$)<>0 THEN PRINT "{2DOWN,5SPAC
     E}BITTE FEHLERMELDUNG BEACHTEN:{2SPACE}
     ",,DS$                                     <H0C>
1070 END                                        <15U>
1080 IF ER=14 THEN 1100                         <QV2>
1090 IF ER=30 THEN RESUME : ELSE PRINT "{DOW
     N,RIGHT}" ERR$(ER): RESUME 1070            <R1V>
1100 PRINT TAB(21) ERR$(ER): SLEEP 2: PRINT
     "{UP}";CHR$(27)"D"                         <0NN>
1110 PRINT "{2UP}": RESUME 530                  <8NU>
1120 TRAP : PRINT "{HOME,4DOWN}" CHR$(27)"@{
     3SPACE}NEUER START MIT 'RUN' !{2DOWN}":
      END                                       <7LP>
```

Listing 9. Ein Boot-Sektor mit 17 Möglichkeiten

Auf die Frage »Text hinter 'Booting'?« kann ein beliebiger Text eingegeben werden (Text = TX$). Wird der Name des zu ladenden Programms eingetippt, erscheint während des Ladens:

"BOOTING (Programmname)..."

Aber dank des bereits durch Zeile 360 in den Tastaturpuffer vorgegebenen Anführungszeichens können auch die bereits oben erwähnten Steuerzeichen eingegeben werden.

Wenn kein Text erwünscht ist: <RETURN> drücken!

Für die richtige Positionierung der Punkte (...) wurde in Zeile 380 für die TAB-Funktion aus Speicherstelle 215 ($D7) der Wert für 40- oder 80-Zeichen-Darstellung abgefragt. Dort steht entweder $00 oder $80 für 40 beziehungsweise 80 Zeichen. Denn: 0:3,2 ergibt TAB(0) und 128:3,2 ergibt TAB(40)!

Zeile 400: Soll ein Maschinen-Programm geladen werden, kann jetzt der Name des Programms eingegeben werden. Kein Maschinen-Programm? <RETURN> drücken!

Wurde ein Name eingegeben (MP$), muß nun die Frage »Autostart?« mit J(a) oder N(ein) beantwortet werden (Zeilen 420 bis 480).

Zeile 470: Die Variable RU ist für die Einsprungadresse zum eventuellen Laden eines Maschinen-Programms wichtig, und ergibt sich aus der Länge des Textes (TX$), der Länge des Maschinenprogramm-Namens (MP$) und dem Wert von JN=19 plus 3 für einen JSR-Befehl, falls Autostart gewählt wurde.

In Zeile 490/500 wird wieder für die PRINT-Anweisung der aktuelle Bildschirm abgefragt. Falls das Maschinen-Programm nicht automatisch gestartet werden soll, geht es von Zeile 490 direkt nach 650 zur Frage nach einem Basic-Programm. Ansonsten muß nun erst noch die Startadresse genannt werden (Zeilen 530 bis 630). Wer die Adresse nicht in HEX-Werten weiß, kann selbstverständlich auch die SYS-Adresse in dezimal angeben; man braucht dann nur N(ein) mit <RETURN> zu bestätigen. Wer unmögliche Werte eingibt, TRAPt in die Fehlerfalle und darf die Eingabe wiederholen.

Die Zeilen 620/630 zerlegen die Adresse in Low- (JL) und High-(JH)Byte.

Nun wird (Zeile 650) nach einem Basic-Programm gefragt. Soll kein Basic-Programm geladen werden, nur <RETURN> drücken. Andernfalls muß der Programmname eingegeben werden (BP$).

Entsprechend der Entscheidung: Autostart – ja oder nein wird der Variablen BS$ der Wert RUN für Laden und Starten oder DLOAD für Laden ohne Starten übergeben (Zeilen 670 bis 710).

Sollte weder ein Text noch ein Programmname eingegeben worden sein, wäre ein Boot-Sektor überflüssig und das Programm wird beendet.

Im Abschnitt der Zeilen 770 bis 830 wird mit GETKEY OK$ angehalten und die Eingaben können in aller Ruhe überprüft werden. Durch X bricht das Programm ab, mit N können laut Zeile 340 nach Löschung aller Variablen (CLR) alle Eingaben erneuert werden.

Ist alles in Ordnung, muß nur noch J gedrückt werden. Nachdem den Variablen CB$ und NB$ die Werte CBM und 00 zugewiesen wurden, läuft der eigentliche Schreibvorgang des Boot-Sektors in den Zeilen 880 bis 1010 ab. In Zeile 1010 wird mit TRAP die Fehlerfalle desaktiviert. Zeile 1020 schafft mit ESCAPE <Klammeraffe> Platz für die Floppy-Meldung DS$.

Fehlermeldungen von der Floppy werden in den Zeilen 1040 bis 1060 und Basic-Fehler in den Zeilen 1080 bis 1120 verarbeitet. (Hans-Georg Schmidt/dm)

Variablen- und Array-Inhalte anzeigen

1): DUMPV 128

Dieses Programm (Listing 10, bitte mit dem neuen Checksummer 128 eingeben) ermöglicht die Ausgabe der Werte der nichtindizierten Variablen eines Basic-Programms.

Nach Start des Ladeprogramms mit RUN erfolgt der Aufruf mit SYS DEC ("0B00"). Es werden für jede Variable deren Name (maximal die ersten beiden Buchstaben), deren Typ (SPACE für Real, % für Integer und $ für String) sowie deren Werte ausgegeben.

2): DUMPA 128

Dieses Programm (Listing 11, bitt ebenfalls mit dem neuen Checksummer 128 eingeben) funktioniert ähnlich wie DUMPV 128, nur daß die Werte der indizierten Variablen (Arrays) eines Basic-Programms ausgegeben werden. Nach Start des Ladeprogramms mit RUN kann es mit SYS DEC

```
110 REM                                        <02B>
120 REM              DUMPV 128                 <028>
130 REM                                        <029>
140 REM PROGRAMM ZUR AUTOMATISCHEN AUSGABE D
    ER NICHTINDIZIERTEN VARIABLEN              <02E>
150 REM EINES BASIC 7.0 PROGRAMMES FUER DEN
    COMMODORE 128                              <02F>
160 REM                                        <02C>
250 SCREENMODE=PEEK(DEC("00D7"))               <PPQ>
260 IF SCREENMODE=0 THEN SP=0: SX=15: SY=10:
     ELSE SP=5: SX=35: SY=10                   <HF6>
270 SCNCLR SP                                  <K5N>
280 COLOR 0,1: COLOR 6,1: COLOR 4,1: COLOR 5
    ,2                                         <CJU>
290 CHAR 1,SX,SY,"DUMPV 128",1                 <IET>
300 REM DATA-ZEILEN LESEN                      <02E>
310 S=0: FOR I=DEC("0B00") TO DEC("0C1D"): R
    EAD X: S=S+X: POKE I,X: NEXT I             <UCC>
320 DATA 169,0,141,0,255,165,47,133,252,165,
    48,133,253,197,50,208,12,165,252           <5OM>
330 DATA 197,49,208,6,169,63,141,0,255,96,16
    0,0,32,230,11,16,6,32,78,11,76             <8GN>
340 DATA 57,11,200,32,230,11,16,6,32,147,11,
    76,57,11,32,115,11,169,13,32,45            <UPQ>
350 DATA 199,24,165,252,105,7,133,252,165,25
    3,105,0,133,253,76,13,11,32,200            <7PC>
360 DATA 11,169,37,32,45,199,169,32,32,45,19
    9,32,230,11,133,100,200,32,230             <860>
370 DATA 11,133,101,162,144,56,32,112,140,32
    ,66,142,32,226,85,96,32,200,11             <E2E>
380 DATA 169,32,32,45,199,169,32,32,45,199,1
    66,252,164,253,232,208,1,200,232           <HAM>
390 DATA 208,1,200,138,32,246,11,76,108,11,3
    2,200,11,169,36,32,45,199,169              <ROI>
400 DATA 32,32,45,199,169,32,32,45,199,32,23
    0,11,133,254,240,27,200,32,230             <NND>
410 DATA 11,72,200,32,230,11,133,251,104,133
    ,250,160,0,32,238,11,32,45,199             <1U7>
420 DATA 200,198,254,208,245,96,169,32,32,45
    ,199,160,0,32,230,11,41,127,32             <E2E>
430 DATA 45,199,200,32,230,11,41,127,208,2,1
    69,32,32,45,199,200,96,165,252             <TOF>
440 DATA 133,250,165,253,133,251,169,250,162
    ,1,32,116,255,96,133,250,132,251           <5LV>
450 DATA 160,4,32,238,11,133,103,136,32,238,
    11,133,102,136,32,238,11,133,101           <4UK>
460 DATA 136,32,238,11,133,104,9,128,133,100
    ,136,32,238,11,133,99,96                   <A4I>
470 SCNCLR SP                                  <N5N>
480 IF S=32209 THEN 510                        <MA5>
490 CHAR 1,SX,SY,"FEHLER IN DATAZEILEN !!!"    <K2M>
500 CHAR 1,SX,SY+2,"DIFFERENZ = "+STR$(32209
    -S)                                        <HUN>
510 END                                        <0NU>
```

Listing 10. »DumpV 128« – Ausgabe nichtindizierter Variablen

("0C1E") aufgerufen werden. Die Ausgabe der Namen und Felder erfolgt analog zur Ausgabe der Variablennamen bei DUMPV 128. Die Werte eines Arrays werden, getrennt durch Kommata, in der Reihenfolge, in der sie im Speicher stehen, ausgegeben.

Da beide Programme verschiedene RAM-Bereiche belegen, können sie gleichzeitig im Speicher stehen und unabhängig voneinander verwendet werden. Die Werte aller Variablen (also einfache und indizierte Variable) können mit SYS DEC ("0DE6") ausgegeben werden, wenn vorher folgendes Programm gestartet wird:

```
10 FOR I=DEC("0DE6") TO DEC("0DEC"): READ X
20 POKE I,X: NEXT I
30 DATA 32,0,11,32,30,12,96
```

(Lothar Glässer/dm)

Aus dem Directory laden

Wird ein Programm mit »DSAVE"NAME[SHIFT+SPACE]:"« gespeichert dann findet man im Directory »"Name":« vor. Es kann nun nach <F3> einfach durch Anfahren mit dem Cursor und <F2> + <RETURN> geladen werden. Mit dem RENAME-Befehl lassen sich natürlich schon vorhandene Programme entsprechend umbenennen.

(Gerhard Seebauer/dm)

```
110 REM                                   <02B>
120 REM           DUMPA 128               <028>
130 REM                                   <029>
140 REM PROGRAMM ZUR AUTOMATISCHEN AUSGABE D
    ER INDIZIERTEN VARIABLEN              <02E>
150 REM EINES BASIC 7.0 PROGRAMMES FUER DEN
    COMMODORE 128                         <02F>
160 REM                                   <02C>
250 SCREENMODE=PEEK(DEC("00D7"))          <PPQ>
260 IF SCREENMODE=0 THEN SP=0: SX=15: SY=10:
     ELSE SP=5: SX=35: SY=10              <HF6>
270 SCNCLR SP                             <K5N>
280 COLOR 0,1: COLOR 6,1: COLOR 4,1: COLOR 5
    ,2                                    <CJU>
290 CHAR 1,SX,SY,"DUMPA 128",1            <J0T>
300 REM DATA-ZEILEN LESEN                 <02E>
310 S=0: FOR I=DEC("0C1E") TO DEC("0DE5"): R
    EAD X: S=S+X: POKE I,X: NEXT I        <8K8>
320 DATA 169,0,141,0,255,165,49,133,252,165,
    50,133,253,197,52,208,12,165,252      <DON>
330 DATA 197,51,208,6,169,63,141,0,255,96,16
    0,0,32,163,13,16,6,32,97,12,76        <22H>
340 DATA 87,12,200,32,163,13,16,6,32,233,12,
    76,87,12,32,171,12,169,13,32,45       <JV0>
350 DATA 199,165,253,76,43,12,32,67,13,169,3
    7,32,45,199,169,32,32,45,199,160      <6NI>
360 DATA 0,32,174,13,133,100,200,32,174,13,1
    33,101,162,144,56,32,112,140,32       <MK0>
370 DATA 66,142,32,226,85,24,165,22,105,2,13
    3,22,165,23,105,0,133,23,197,253      <5LK>
380 DATA 240,4,176,16,144,6,165,22,197,252,1
    76,8,169,44,32,45,199,76,110,12       <N7T>
390 DATA 96,32,67,13,169,32,32,45,199,169,32
    ,32,45,199,165,22,164,23,32,190       <JTJ>
400 DATA 13,32,66,142,32,226,85,24,165,22,10
    5,5,133,22,165,23,105,0,133,23        <DVC>
410 DATA 197,253,240,4,176,16,144,6,165,22,1
    97,252,176,8,169,44,32,45,199         <892>
420 DATA 76,184,12,96,32,67,13,169,36,32,45,
    199,169,32,32,45,199,169,32,32        <3G5>
430 DATA 45,199,160,0,32,174,13,133,254,240,
    27,200,32,174,13,72,200,32,174        <CQK>
440 DATA 13,133,251,104,133,250,160,0,32,182
    ,13,32,45,199,200,198,254,208         <9JH>
450 DATA 245,24,165,22,105,3,133,22,165,23,1
    05,0,133,23,197,253,240,4,176         <EK6>
460 DATA 16,144,6,165,22,197,252,176,8,169,4
    4,32,45,199,76,246,12,96,169,32       <F9F>
470 DATA 32,45,199,160,0,32,163,13,41,127,32
    ,45,199,200,32,163,13,41,127,208      <7DH>
480 DATA 2,169,32,32,45,199,160,4,24,165,252
    ,105,5,133,22,165,253,105,0,133       <76H>
490 DATA 23,32,163,13,24,101,22,133,22,169,0
    ,101,23,133,23,32,163,13,24,101       <LR2>
500 DATA 22,133,22,169,0,101,23,133,23,160,2
    ,32,163,13,24,101,252,8,133,254       <HFV>
510 DATA 200,32,163,13,40,101,253,133,253,16
    5,254,133,252,96,165,252,133,250      <2BL>
520 DATA 165,253,133,251,76,182,13,165,22,13
    3,250,165,23,133,251,169,250,162      <KDR>
530 DATA 1,32,116,255,96,133,250,132,251,160
    ,4,32,182,13,133,103,136,32,182       <Q76>
540 DATA 13,133,102,136,32,182,13,133,101,13
    6,32,182,13,133,104,9,128,133         <EA3>
550 DATA 100,136,32,182,13,133,99,96      <RM1>
560 SCNCLR SP                             <NTN>
570 IF S=47090 THEN 600                   <ISF>
580 CHAR 1,SX,SY,"FEHLER IN DATAZEILEN !!!"  <K4M>
590 CHAR 1,SX,SY+2,"DIFFERENZ = "+STR$(47090
    -S)                                   <DNR>
600 END                                   <0M6>
```

Listing 11. »DumpA 128« – Ausgabe indizierter Variablen

Den Zehnerblock sinnvoll belegen

Das nächste Programm (Listing 12) demonstriert den Gebrauch der Interruptroutine und der Tastaturbelegung des C128 anhand einer Neubelegung des Zehnerblocks bei gedrückter SHIFT-Taste.

Will man DATA-Werte eingeben, so kommt einem doch der Zehnerblock des C128 sehr gelegen. Um diesen Vorteil auch bei der Assemblerprogrammierung nutzen zu können, fehlt es jedoch an den Buchstaben A bis F. Weiterhin ist zu bemängeln, daß es beim eingebauten Monitor zu Problemen kommt, wenn man einen Teil disassembliert und anschließend verändern will. Stehen hinter dem neu Eingegebenen noch Zeichen, so akzeptiert der Monitor die Eingabe nicht und gibt statt dessen ein mit der Zeit nervendes Fragezeichen aus. Eine Verbesserung der Belegung der Zehnerblock-Tasten, besonders, wenn man mit dem Monitor arbeitet, bewirkt das Programm »10'ER BLOCKER«. Dabei wird lediglich die SHIFT-Version der 14 Zehnerblock-Tasten neu belegt. Die Belegung ohne SHIFT bleibt gleich (Bild 1 zeigt die neue Belegung des Zehnerblocks, gedrückt mit SHIFT).

Untergebracht sind die Buchstaben A bis F, das Komma sowie die folgenden Zeichen: »$«, »%«, »&«, »/« und »*«. Die Null-Taste ergibt, gedrückt mit SHIFT, zwei Nullen (00). Der RETURN-Taste wird in diesem Fall ein Doppelpunkt vorgeschoben. Befindet man sich im Monitor und verändert ein Assemblerlisting, so werden, falls man SHIFT und RETURN gleichzeitig drückt, die Zeichen hinter dem Cursor nicht beachtet.

```
15 REM  * * *  10'ER BLOCKER  * * *       <01B>
20 REM                                    <01A>
40 REM  NEUBELEGUNG DER SHIFT/10'ER       <01C>
50 REM  BLOCK - TASTEN ; DURCH VER-       <01D>
60 REM  AENDERN DER INTERRUPTROUTI-       <01E>
70 REM  NE UND DER TASTATURBELEGUNG       <01F>
80 REM                                    <010>
100 BANK 15: POKE 53280,0: POKE 53281,0   <MCK>
110 PRINT CHR$(147);CHR$(30)              <DLE>
120 PRINT "AN WELCHE ADRESSE (IN HEX) SOLL D
    IE": PRINT                            <KJB>
130 PRINT "ROUTINE? PLATZBEDARF:1 BLOCK (256
     BYTES)": PRINT                       <72J>
140 PRINT : INPUT "ADRESSE ($)";A$: FAST  <LEJ>
150 A=DEC(A$): IF A=0 THEN A=5888: REM $1700 <7ER>
155 REM 1.SCHLEIFE FUER DATA-ZAHLEN       <02D>
160 FOR N=A TO A+139: READ Z: POKE N,Z: S=S+
    Z: NEXT                               <11T>
170 IF S<>17017 THEN PRINT "FEHLER IN DATA'S
    !!!": STOP                            <B3P>
175 REM 2.SCHLEIFE FUER DATA-ZEICHEN      <02F>
180 FOR N=A+140 TO A+165: READ Z$: Z=VAL(Z$)
    : IF Z=0 THEN Z=ASC(Z$)               <VFH>
190 POKE N,Z: NEXT                        <51L>
200 PRINT CHR$(147);"INITIALISIEREN MIT:": P
    RINT : PRINT : PRINT                  <7BK>
210 PRINT "BANK15:SYS"A;CHR$(145);CHR$(145);
    CHR$(145)                             <J92>
215 REM ANPASSEN AN DIE ADRESSE A         <02F>
220 L=(A+166) AND 255: H=INT((A+166)/256) <912>
230 POKE(A+2) ,(A+13) AND 255             <UI2>
240 POKE(A+7) ,INT((A+13)/256)            <QOD>
250 POKE(A+67),L: POKE(A+68),H            <GMU>
260 POKE(A+80),(A+148) AND 255            <13T>
270 POKE(A+81),INT((A+148)/256)           <6U0>
280 POKE(A+83),(A+231) AND 255            <I5T>
290 POKE(A+84),INT((A+231)/256)           <QLA>
300 POKE(A+91),L: POKE(A+93),H            <J46>
310 SLOW : END                            <L06>
320 DATA 120,169,13,141,20,3,169,23,141,21,3
    ,88,96,216,32,36                      <UDG>
330 DATA 192,144,18,32,248,245,32,208,238,17
    3,13,220,173,4,10,74                  <DT3>
340 DATA 144,3,32,6,64,173,65,3,201,240,144,
    56,174,0,255,169                      <UGV>
350 DATA 15,141,0,255,172,64,3,132,254,172,6
    5,3,132,255,160,0                     <1JH>
360 DATA 177,254,153,166,23,200,192,89,208,2
    46,142,0,255,160,0,185                <KML>
370 DATA 148,23,153,231,23,200,192,18,208,24
    5,162,166,160,23,142,64               <QK2>
380 DATA 3,140,65,3,165,211,106,144,22,166,2
    08,240,18,189,73,3                    <72M>
390 DATA 201,48,208,14,236,32,10,176,6,232,1
    57,73,3,134,208,76                    <2E1>
400 DATA 51,255,201,141,208,249,236,32,10,17
    6,244,169,58,157,73,3                 <4SJ>
410 DATA 169,13,208,229,"%","E",24,"B","D","
    $","A",27,"*","/"                     <A9Q>
420 DATA 10,141,"F","&","C",8,48,","      <IR8>
```

Listing 12. Neubelegung der geSHIFTeten Tasten des Zehnerblocks

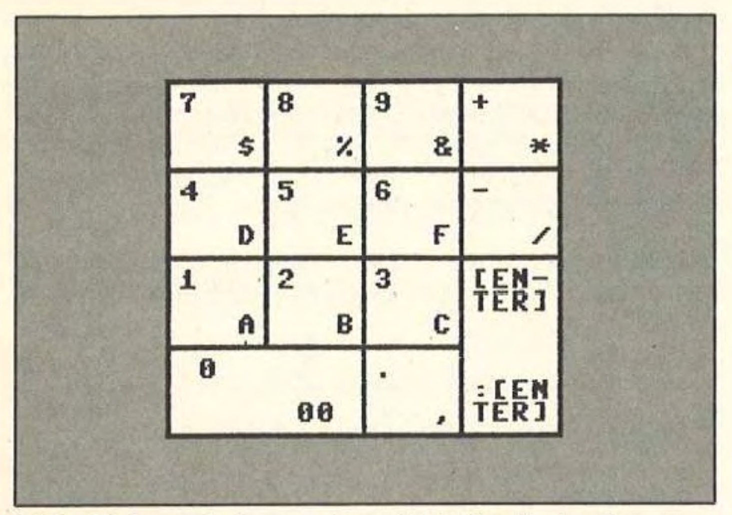


Bild 1. Die Tastenbelegung der geSHIFTeten Tasten des Zehnerblocks

Die Tastaturbelegung

Zunächst muß man wissen, daß der C128 mehrere Tastatur-Decodiertabellen besitzt. Diese Tabellen sind für die verschiedenen Tastenkombinationen mit SHIFT-, CONTROL- und COMMODORE-Tasten vorhanden. Außerdem gibt es weitere Tabellen für die DIN-Belegung, die sich von der ASCII-Version unterscheidet.

Die Tabellen, die uns interessieren, sind die, die in Verbindung mit der SHIFT-Taste gelten. Der Vektor, der auf diese Tabelle zeigt, liegt bei $340/$341. Man transferiert also die Tabelle, auf die dieser Vektor zeigt ($FAD9 bis $FB31), ins RAM, verändert die entsprechenden Werte und verbiegt den Vektor auf die neue Tabelle. Dies funktioniert jedoch nur, bis die ASCII/DIN-Taste gedrückt wird, da dann der Vektor wieder ins ROM auf die DIN-Decodiertabelle ($FD82 bis $FDDA) gestellt wird.

Auch eine Belegung der Null- und RETURN-Taste mit zwei Zeichen gleichzeitig ist so nicht zu realisieren. Die einzige vernünftige Lösung scheint über den Interrupt zu laufen.

Der Interrupt

Beim C128 gibt es, ähnlich wie beim C64, einen Interrupt-Vektor. Dieser Vektor liegt bei $314/$315 und zeigt auf die Routine bei $FA65. Diese Routine geht bis $FA7C und springt anschließend nach $FF33. Will man eigene Routinen in den System-Interrupt einbauen, so kopiert man zunächst diese ROM-Routine ins RAM (etwa nach $1300) und hängt die eigene Routine hinten an. Am Ende der eigenen Routine springt man nach $FF33, um den Interrupt mit dem Wiederherstellen der Register zu beenden. Aktiviert wird diese neue Routine, indem man den Interrupt-Vektor auf ihren Anfang lenkt.

Das Programm

Der Hauptteil dieses Programms läuft im Interrupt ab. Nachdem der Interrupt-Vektor ($314/$315) durch SYS DEC ("1700") auf die Hauptroutine bei $1700 gestellt wurde, werden dort zunächst die gleichen Unterroutinen aufgerufen, wie es die Interrupt-Routine bei $FA65 auch machen würde. Anschließend wird kontrolliert, ob der Zeiger auf die Tastatur-Decodiertabelle ins ROM zeigt. Ist dies der Fall, wird diese Tabelle nach $17A6 kopiert und die Neuerungen im Zehnerblock-Bereich vorgenommen. Nun muß nur noch der Vektor bei $340/$341 auf die neue Tabelle gestellt werden.

Als nächstes prüft das Programm, ob die SHIFT-Taste gedrückt ist. Dazu existiert in der Zeropage ein Flag. Ist die SHIFT-Taste gedrückt, so ist das 0. Bit von $D3 gesetzt. Trifft dies zu, so wird das letzte Zeichen aus dem Tastaturpuffer gelesen (Anzahl der Zeichen im Tastaturpuffer in $D0, Tastaturpuffer ab $349, maximale Länge des Tastaturpuffers bei $A20). Handelt es sich bei dem Zeichen um eine Null, so wird diese nochmals in den Tastaturpuffer geschoben. Ist dies jedoch die RETURN-Taste, so wird zunächst ein Doppelpunkt und anschließend ein RETURN in den Tastaturpuffer geschrieben.

Der Basic-Lader (Listing 12, bitte mit dem neuen Checksummer 128 eingeben) kann das 255 Byte lange Programm an jede beliebige Stelle speichern. Es empfehlen sich folgende Bereiche für das Maschinenprogramm: $0B00 bis $0DFF (Kassetten- und RS232-Ein- und Ausgabepuffer)

```
************** Interruptvektor veraendern
1700 sei         ; Interrupt verbieten
1701 lda #$0d    ; Interruptvektor bei $0314/$0315
1703 sta $0314   ; auf $170d stellen
1706 lda #$17    ; (normal: $fa65)
1708 sta $0315   ;
170b cli         ; Interrupt wieder erlauben
170c rts         ; und zurueck zur Einsprungsadresse

************** Interrupt Einsprung
170d cld         ;
170e jsr $c024   ;
1711 bcc $1725   ;
1713 jsr $f5f8   ; Kopie der ROM-Interruptroutine
1716 jsr $eed0   ; von $fa65 bis $fa7c
1719 lda $dc0d   ;
171c lda $0a04   ;
171f lsr         ;
1720 bcc $1725   ;
1722 jsr $4006   ;

************** Neuer Teil der Interruptroutine
1725 lda $0341   ; Hi-Byte der aktuellen Tast.dec.tabelle
1728 cmp #$f0    ; zeigt in den ROM-Bereich ? ($fa o. $fd)
172a bcc $1764   ; Nein: weiter                (ASCII  DIN)
172c ldx $ff00   ; aktuelles Konfig.-register speichern
172f lda #$0f    ; neue Konfiguration:
1731 sta $ff00   ; Kernal-ROM einschalten
1734 ldy $0340   ; aktuellen Zeiger auf Tastaturdecodier-
1737 sty $fe     ; tabelle 2 in die Zeropage
1739 ldy $0341   ; ($0340/$0341) nach ($fe/$ff)
173c sty $ff     ;
173e ldy #$00    ; Schleife zum transferieren der
1740 lda ($fe),y ; aktuellen Tastaturdecodiertabelle 2
1742 sta $17a6,y ; (ASCII oder DIN) an das Ende dieser
1745 iny         ; Routine ($17a6)
1746 cpy #$59    ; laenge der Tabelle
1748 bne $1740   ;
174a stx $ff00   ; alte Speicherkonfig. wieder herstellen
174d ldy #$00    ;
174f lda $1794,y ; Schleife zum Einbauen der neuen
1752 sta $17e7,y ; Zeichen in die neue Tastaturdecodier-
1755 iny         ; tabelle 2
1756 cpy #$12    ;
1758 bne $174f   ;
175a ldx #$a6    ; Zeiger ($0340/$0341) auf neue
175c ldy #$17    ; Tastaturdec.tabelle ($17a6)
175e stx $0340   ;
1761 sty $0341   ;
************** Pruefen auf SHIFT/RETURN o. -/"0"
1764 lda $d3     ; Pruefen auf SHIFT-Taste
1766 ror         ; (0.Bit gesetzt=SHIFT gedrueckt)
1767 bcc $177f   ; nicht gedrueckt: Ende
1769 ldx $d0     ; Anzahl Zeichen im Tastaturpuffer
176b beq $177f   ; leer: Ende
176d lda $0349,x ; Zeichen aus Tastaturpuffer holen
1770 cmp #$30    ; gleich ASCII "0" ?
1772 bne $1782   ; nein: Testen auf RETURN
1774 cpx $0a20   ; Tastaturpuffer voll ?
1777 bcs $177f   ; ja: Ende
1779 inx         ; Anzahl der Zeichen im Puffer um 1 erhoehen
177a sta $0349,x ; Zeichen in Tastaturpuffer schreiben
177d stx $d0     ; Anzahl der Zeichen im Puffer abspeichern
177f jmp $ff33   ; ****** Ende ****** Ruecksprung zum ROM-IRQ
1782 cmp #$8d    ; Zeichen = SHIFT/RETURN ?
1784 bne $177f   ; Nein: Ende
1786 cpx $0a20   ; Noch Platz im Tastaturpuffer ?
1789 bcs $177f   ; Nein: Ende
178b lda #$3a    ; ASCII ":"
178d sta $0349,x ; ":" in den Tastaturpuffer
1790 lda #$0d    ; ASCII RETURN
1792 bne $1779   ; Unbedingter Sprung

 1794 .byte 25 45 18 42 44 24 41 1b ; neue Zeichen der Tastatur-
 179c .byte 2a 2f 0a 8d 46 26 43 08 ; decodiertabelle 2
 17a4 .byte 30 2c                   ;

 17a6 - 17ff     ; Platz fuer neue Tastaturdecodiertabelle
```

Listing 13. Der kommentierte Quellcode zum 10'er-Blocker

sowie das freie RAM von $1300 bis $1F77. Gibt man keine Adresse an, so wird es automatisch in den Bereich von $1700 bis $17FF geschrieben.

Ebenfalls ist es möglich, die geSHIFTeten Zehnerblock-Tasten beliebig zu belegen. Dazu müssen lediglich in den DATA-Zeilen 410 und 420 des Listings 12 die entsprechenden ASCII-Zeichen verändert werden.

Beispiel: Will man die SHIFT-Kombination der Taste »9« im Zehnerblock mit dem Zeichen »#« anstelle des »&« belegen, so braucht man nur in der Zeile 420 das »&«-Zeichen durch das »#« zu ersetzen.

Für Maschinensprache-Experten ist in Listing 13 der kommentierte Quell-Code abgedruckt. (Olaf Pfeiffer/dm)

Fehlermeldungen, die nicht im Handbuch stehen

Auch das überraschend dicke Commodore-Handbuch zum C128 vermag nicht alles zu beantworten. So hat wohl schon so mancher, dem zum Beispiel die Fehlermeldung UNRESOLVED REFERENCE ERROR unterkam, nachgeschlagen, gesucht und doch nicht gefunden...

Insgesamt existieren fünf Fehlermeldungen, die nicht im Handbuch zu finden sind und auch sonst nirgendwo erwähnt werden. Dies sind die Meldungen mit den Nummern 37 bis 41, die Sie auch mit der Funktion ERR$(ER) ansehen können. Hier erfahren Sie nun endlich, was die Meldungen im einzelnen bedeuten:

ER 37: BEND NOT FOUND

Dieser Fehler tritt auf, wenn der Interpreter zu einem BEGIN kein BEND findet. Hat man zum Beispiel eine Befehlssequenz wie

```
IF (Bedingung) THEN BEGIN
```

und die Bedingung ist nicht erfüllt, so sucht der Interpreter nach dem BEND. Ist im nachfolgenden Programm vor dem nächsten BEGIN kein BEND mehr vertreten, so erhalten Sie die oben genannte Fehlermeldung mit der Zeilennummer der letzten Basic-Zeile. Drückt man die HELP-Taste, so wird die ganze Zeile als falsch angegeben. Sehr ärgerlich kann der Fehler sein, wenn in einem Programm mehrere BEGIN und BEND existieren. Dann müssen Sie Ihr Programm von Hand durchsuchen. Als Beispiel sehen Sie Listing 14, das die Wurzel einer Zahl berechnen soll. Die Wurzel kann aber nur von einer positiven Zahl gezogen werden. Also muß bei negativen Zahlen auf die Berechnung verzichtet werden. In diesem Fall soll zum END verzweigt werden. Ihnen ist sicher schon aufgefallen, daß in Zeile 50 das BEND fehlt. Lassen Sie das Programm einmal laufen. Geben Sie eine positive Zahl ein, so arbeitet die Routine richtig. Ist Ihre Zahl negativ, so erhalten Sie ein BEND NOT FOUND ERROR IN 540, also der letzten REM-Zeile. Fügen Sie das BEND ein, so arbeitet das Programm ordnungsgemäß.

Geben Sie nun einmal

```
RENUMBER 10000,10000
```

ein. Damit sind wir schon beim nächsten Fehler.

ER 38: LINE NUMBER TO LARGE

Auf den ersten Blick könnte man vermuten, daß dieser Fehler auftritt, wenn man eine zu große Zeilennummer eingibt. Dies ist aber nicht der Fall. Der Interpreter würde hier einen SYNTAX ERROR ausgeben. Die oben genannte Fehlermeldung tritt beim RENUMBER-Befehl auf, wenn bei der Umnumerierung Zeilennummern auftreten würden, die zu hoch sind. Die höchste erlaubte Zeilennummer ist 63999. Falls Sie bei Listing 14

```
RENUMBER 10000,10000
```

eingeben, so würde in der siebten Programmzeile schon die Zeilennummer 70000 auftreten.

ER 39: UNRESOLVED REFERENCE

Diese Fehlermeldung bezieht sich auch nur auf den RENUMBER-Befehl. Es handelt sich hierbei um das Gegenstück zum UNDEF'D STATEMENT ERROR. Es wird die Nummer der Zeile ausgegeben, in der sich ein Sprungbefehl befindet (GOTO, GOSUB), der sich auf eine Zeile bezieht, die nicht existiert. Fügen Sie beispielsweise

```
55 GOSUB 1000
```

in Listing 14 ein. Wollen Sie das Programm jetzt umnumerieren, erhalten Sie einen UNRESOLVED REFERENCE ERROR IN 55.

```
10 INPUT "{2SPACE}GEBEN SIE EINE ZAHL EIN ";
   A                                          <G4K>
20 IF A>=0 THEN BEGIN                         <JDU>
30 W=SQR(A)                                   <FSL>
40 PRINT "{2SPACE}DIE WURZEL VON";A;"IST";W   <GT6>
60 END                                        <0CM>
70 :                                          <023>
80 :                                          <02T>
500 REM  WENN DIE ZAHL GROESSER ODER          <022>
510 REM  GLEICH 0 IST, SOLL DIE WURZEL        <023>
520 REM  BERECHNET WERDEN. IST DIE ZAHL       <020>
530 REM  KLEINER 0 SO SOLL NICHTS GE-         <021>
540 REM  MACHT WERDEN.                        <026>
```

Listing 14. Ein Versuchs-Programm, um Fehlermeldungen zu erklären (siehe Text)

Allerdings kann man sich diesen Fehler zunutze machen. Haben Sie beispielsweise ein langes Basic-Programm, aus dem Sie jetzt eine Zeile löschen müssen, besteht die Gefahr, daß diese Zeile über GOTO oder GOSUB angesprungen werden könnte. Lassen Sie das Programm laufen, so erhalten Sie einen UNDEF'D STATEMENT ERROR. Angenommen, Listing 14 enthielte ein Unterprogramm ab Zeile 1000, das Sie jetzt gelöscht haben. Sie wissen aber nicht, von wo es angesprungen wurde. Eigentlich hätte man im komfortableren Basic 7.0 einen FIND-Befehl erwarten können. Leider ist das nicht der Fall. Um nun die Zeilen zu finden, die die Zeile 1000 anspringen, fügen Sie an das Listing 14 beispielsweise die Zeile

```
63000 GOTO 63001
```

an, wobei die Zeile 63001 nicht existieren darf. Geben Sie jetzt einfach RENUMBER ein. Daraufhin erhalten Sie einen UNRESOLVED REFERENCE ERROR IN 55. Drücken Sie die HELP-Taste, so haben Sie die erste Stelle gefunden, von der aus das Unterprogramm angesprungen wurde. Ändern Sie diese Zeile und geben Sie wieder RENUMBER ein, bis Sie einen UNRESOLVED REFERENCE ERROR IN 63001 erhalten. Dann sind alle Stellen korrigiert. Vielleicht haben Sie sich schon gefragt, was die Zeile 63000 bewirken soll? Nun, solange das Programm nicht einwandfrei ist, beginnt der Interpreter nicht mit der eigentlichen Umnumerierung.

Die Fehlermeldungen 40 und 41 (UNIMPLEMENTED COMMAND ERROR und FILE READ ERROR) konnten noch nicht genau entschlüsselt werden, da die nähere Funktion vom Namen her vielleicht erkennbar ist, aber noch keine stichhaltigen Beweise für Vermutungen vorliegen. Wenn Sie diese Fehlermeldungen schon entlarvt haben, sind wir jederzeit für Tips dankbar. (Thomas Hansch/dm)

Lissajous-Figuren

Ein kleines Programm, das das Grundgerüst zum Berechnen und Zeichnen von Lissajous-Figuren ermöglicht (Listing 15, bitte mit dem neuen Checksummer 128 eingeben). Durch Verändern verschiedener Werte (siehe REM-Zeilen) können Sie alle möglichen Figuren erzeugen.

(Matthias Jäger/dm)

```
10 GRAPHIC 1,1                                   <LUI>
20 WIDTH 2                                       <32H>
30 X0=160: Y0=100: A=150: B=95                   <QOU>
40 D=5: N=3: M=4: T0=π/180                       <CQ3>
50 COLOR 0,1: COLOR 4,2: COLOR 1,2               <LSI>
60 FOR W=0 TO 180*D STEP D                       <NFK>
70 T=W*T0                                        <3HC>
80 X=X0+A*COS(N*T)                               <JJ6>
90 Y=Y0+B*SIN(M*T)                               <D3Q>
100 IF T<>0 THEN DRAW 1,X1,Y1 TO X,Y             <PU2>
110 X1=X: Y1=Y                                   <D57>
120 NEXT W                                       <2LB>
130 COLOR 1,3                                    <LS7>
140 PAINT 1,0,0                                  <LPD>
150 REM FIGUR 1 - D=5  , N=3 , M=4               <02F>
160 REM FIGUR 2 - D=77 , N=1 , M=2               <02C>
170 REM FIGUR 3 - D=77 , N=2 , M=1               <02D>
180 REM FIGUR 4 - D=77 , N=2 , M=3               <022>
190 REM FIGUR 5 - D=77 , N=3 , M=2               <023>
200 REM FIGUR 6 - D=94 , N=1 , M=1               <02C>
210 REM FIGUR 7 - D=94 , N=2 , M=3               <02D>
220 REM FIGUR 8 - D=94 , N=1 , M=2               <02E>
230 REM FIGUR 9 - D=94 , N=1 , M=3               <02F>
```

Listing 15. Mit diesem kleinen Programm können Sie bezaubernde Lissajous-Figuren erzeugen. Bitte benutzen Sie zur Eingabe den Checksummer 128 auf Seite 122.

Noch ein paar PEEKs und POKEs

POKE 160,255	– Zurückstellen der internen Uhr (TI$ = 000000)
PEEK (199)	– zeigt an, in welcher BANK ein File abgelegt ist
POKE 216,0	– ersetzt GRAPHIC 0,1
POKE 216,1	– ersetzt GRAPHIC 1,1
POKE 216,64	– ersetzt GRAPHIC 2,1
POKE 216,128	– ersetzt GRAPHIC 3,1
POKE 216,254	– ersetzt GRAPHIC 4,1
POKE 241,X	– Ändern der Schriftfarbe (X = 0 bis 15)
POKE 243,1	– Einschalten des Revers-Modus
POKE 243,0	– Ausschalten des Revers-Modus
POKE 246,255	– Aktivieren des automatischen Einfügemodus (ESC-A)
POKE 246,1	– Desaktivieren des automatischen Einfügemodus (ESC-C)
POKE 247,64	– Schaltet NO-SCROLL-Taste ab
POKE 247,0	– NO-SCROLL-Taste wieder normal
POKE 247,255	– Unterbinden der Groß-/Kleinschrift-Umschaltung mit COMMODORE/SHIFT
POKE 248,254	– Der Bildschirm rollt sich beim LISTen auf (ESC-M)
POKE 249,255	– Schaltet Klingelzeichen (CTRL-G) ab
POKE 249,0	– Schaltet das Klingelzeichen wieder ein

POKE 774,102:POKE 775,224
– Listschutz

POKE 774,81 :POKE 775,81
– Hebt den Listschutz wieder auf

POKE 820,189	– ESC wird im Quote-Modus sichtbar
POKE 820,185	– ESC reagiert wieder normal
POKE 2594,0	– Keine Tasten-Wiederholung bis auf DEL, INST und CRSR
POKE 2594,64	– Alle Tasten haben Auto-Repeat
POKE 2594,128	– Normalzustand Tasten-Repeat
POKE 2598,64	– bringt den Cursor zum Stillstand
POKE 2598,0	– Cursor blinkt wieder
POKE 2603,128	– Fester Cursor im 80-Zeichen-Modus
POKE 2603,96	– Blinkender Cursor im 80-Zeichen-Modus
POKE 2603,32	– Abgeschalteter Cursor im 80-Zeichen-Modus
POKE 2603,64	– Schnell blinkender Cursor im 80-Zeichen-Modus
SYS 45096	– Monitor wird aktiviert

(Michael Bartels/Helmut Schottmüller/dm)

Laufschrift mit Sprites

Diese auf Spritebasis arbeitende Laufschrift, die jeder Anwender in seine Programme einbauen kann, läßt sich ohne viel Aufwand realisieren. Durch die Interruptsteuerung der Sprites wird der Programmablauf nicht beeinflußt. Dazu ist nur folgendes kleine Programm nötig:

```
10 INPUT"FARBNUMMER (1-16)";F
20 INPUT"GESCHWINDIGKEIT (1-6)";G
30 INPUT"TEXT (BIS 24 ZEICHEN)";I$:SCNCLR
40 GRAPHIC 1,1:CHAR 1,0,0,I$
50 SSHAPE A$,0,0,16,20:SPRSAV A$,1
60 SSHAPE B$,23,0,39,20:SPRSAV B$,2
70 SSHAPE C$,47,0,63,20:SPRSAV C$,3
80 SSHAPE D$,71,0,87,20:SPRSAV D$,4
90 SSHAPE E$,95,0,111,20:SPRSAV E$,5
100 SSHAPE F$,119,0,135,20:SPRSAV F$,6
110 SSHAPE G$,143,0,159,20:SPRSAV G$,7
120 SSHAPE H$,167,0,183,20:SPRSAV H$,8
130 SCNCLR:B=0:FOR A=1TO8:SPRITE A,1,F,1,1,0:B=B+40
140 MOVSPR A,B,51:MOVSPR A,270#G:NEXT
```

(Karl-Heinz Montag/dm)

Bildschirmhardcopy 80-Zeichen-Schirm

Der Video-Controller für den 80-Zeichen-Schirm verwaltet einen eigenen 16 KByte großen Speicher, der außerhalb des normalen Adreßbereiches des Prozessors liegt. Das bedeutet, daß der Video-Speicher nur indirekt angesprochen werden kann, hat aber den Vorteil, daß er keinen Arbeitsspeicher belegt (normalerweise liegt der Bildschirmspeicher innerhalb des VDC-RAMs von $0000 bis $07CF).

Diese kleine Routine erlaubt eine Hardcopy des 80-Zeichen-Bildschirms.

Arnulf Bechmann/dm)

```
10 CLOSE 1:OPEN 4,4,1
20 A=DEC("D0600"):D=A+1
30 FOR J=0 TO 1999 STEP 80
40 FOR Y=0 TO 79
50 BY=J+Y:HB=INT(BY/256):LB=BY*256
60 POKE A,18:POKE D,HB
70 POKE A,19:POKE D,LB
80 POKE A,31:B=PEEK(D)
90 IF B=>1 AND B<=26 THEN B=B+96
100 PRINT#,CHR$(B)
110 NEXT Y,J
120 PRINT#4:CLOSE4:END
```



Tips und Tricks für Basic-Programmierer

Mit den folgenden kleinen Programmen geben wir Ihnen einige Hilfsmittel an die Hand, die Sie in Ihre Programme einbinden können, um sie schneller und effektiver zu machen.

Vielleicht geht es Ihnen wie mir. Sie kommen morgens nicht aus Ihrem Bett, verfluchen jeden Tag aufs neue Ihren Wecker, oder verbrauchen die Nerven Ihres Partners, der verzweifelt versucht, Sie zu wecken. Falls dies der Fall sein sollte, haben Sie sicher schon einmal versucht, sich durch Ihren Computer wecken zu lassen. Sollten Sie zu diesem Zweck die Variable TI$ (die sich ja geradezu aufdrängt) verwendet haben, so würden Sie zwar an diesem Morgen geweckt, jedoch sicher nicht zu dem Zeitpunkt, den Sie sich vorgestellt hatten. Die Variable TI$ hat leider die unangenehme Eigenschaft, eine recht ungenaue Uhr zu sein. Der maximale Fehler liegt bei 30 Minuten pro Tag. Aber zum Glück hat unser Computer, wie so oft, auch hierfür eine Lösung parat. Es handelt sich hierbei um die beiden CIAs (Complex Interface Adapter), welche je eine Echtzeituhr enthalten. Diese Echtzeituhr erhält ihren Takt aus der Netzfrequenz, die Variable TI$ hingegen wird von der Interruptroutine versorgt. Dieser Tatsache verdanken wir es, daß die CIA-Uhr eine so große Langzeitgenauigkeit vorweisen kann. Die Echtzeituhr hält aber noch eine weitere Überraschung für uns bereit. Es besteht die Möglichkeit, eine Alarmzeit anzugeben. Sobald diese Zeit erreicht wird, löst das CIA einen IRQ (Interrupt, beziehungsweise CIA 2 einen NMI = unmaskierter Interrupt) aus, aber dazu später noch mehr. Wie die Register der CIA-Bausteine belegt sind, können Sie aus Tabelle 1 entnehmen. Die Echtzeituhr belegt die Register 8 bis 11.

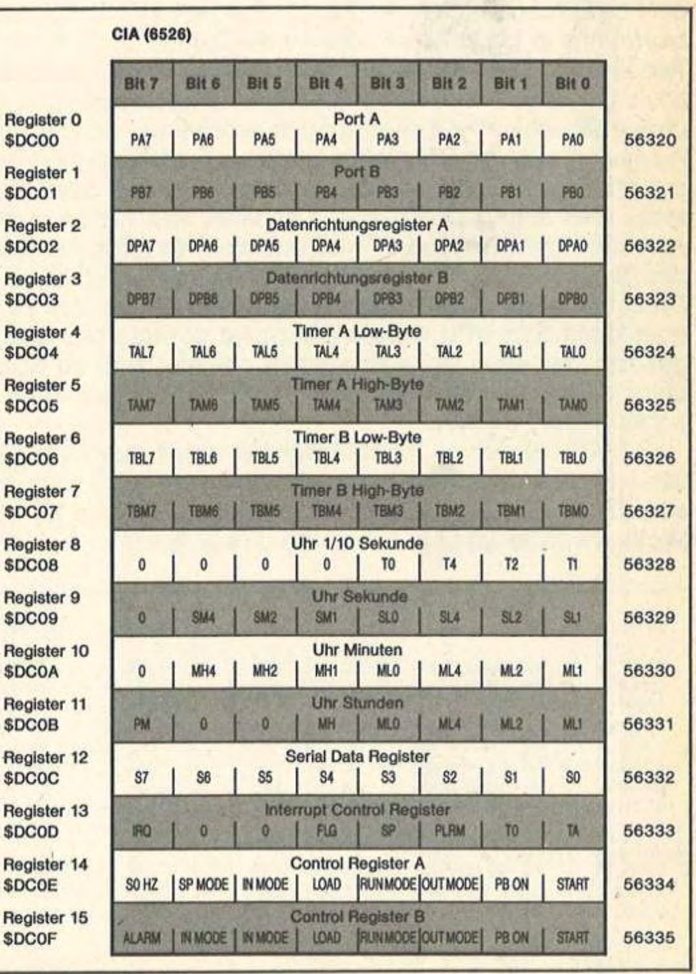
Die Zeit starten

Die CIA-Uhren sind im BCD-Format organisiert. Dadurch sparen wir uns zwar in Maschinensprache das lästige Umrechnen, von Basic aus ist die Umwandlung leider nicht zu umgehen. Im BCD-Format werden Zahlen zwar immer noch Bitweise gespeichert, jedoch jede Ziffer einzeln. Mit 8 Bit lassen sich also zwei Ziffern (2 mal 4 Bit, 00 bis 99) darstellen. Um nun eine Zahl, bestehend aus zwei Ziffern, in das BCD-Format umzuwandeln, wird die erste Ziffer (Wertigkeit 10) mit 16 multipliziert und zu der zweiten Ziffer (Wertigkeit 1) addiert.

Aber nun wieder zu unseren CIAs. Die Basisadresse des CIA 1 ist 56320 (CIA 2 = 56576). Eine kurze Registerbeschreibung entnehmen Sie bitte Bild 1. Zum Stellen der Uhr werden einfach die entsprechenden BCD-Werte in die jeweiligen Register geschrieben. Sobald Sie das $^1/_{10}$ Sekunden-Register überschreiben, startet die Uhr (überschreiben Sie jedoch die Stundenregister, stoppt sie). An Bit 7 in Register 11 können Sie außerdem erkennen, ob es sich um eine AM- oder PM-Zeit handelt (AM = Vormittag, PM = Nachmittag). Bitte vergessen Sie nicht, Bit 7 in Register 14 zu setzen (Netzfrequenz 50 HZ (60 HZ)). Für den Fall, daß Sie Ihre Uhr auch lesen möchten (soll ja vorkommen), bietet das CIA eine weitere Besonderheit. Sobald Sie eines der Register lesen, wird die komplette Uhrzeit in einen Zwischenspeicher übernommen. Durch diesen Kniff können Sie in aller Ruhe die Daten auslesen, ohne daß eine Veränderung des Zwischenspeichers eintritt. Sobald Register 8 angesprochen wird, gibt das CIA den Speicher wieder frei. In Listing 1 wartet ein Basic-Programm auf Sie, mit welchem Sie die Zeit stellen und ablesen können (Start mit RUN, um sie zu stellen und RUN 200, um sie zu stoppen). Listing 2 enthält eine kurze Assemblerroutine, die in die Interruptroutine eingefügt wird und Ihnen die Uhrzeit laufend auf dem Bildschirm ausgibt. Die eigentliche Assemblerroutine liegt bei Adresse 830. Um das Programm zu aktivieren, befindet sich bei Adresse 900 ein Programmteil, welcher den Interruptvektor auf das Hauptprogramm legt (SYS 900 startet die Anzeige der Uhr).

Der Wecker im Computer

Neben einer Uhr hat das CIA natürlich noch weitere Aufgaben. Es löst zum Beispiel das Interruptsignal aus, durch welches die bereits erwähnte Interruptroutine angesprungen wird. Dieses Ereignis tritt bei Unterlauf des Timers (die CIAs haben neben der Uhr noch zwei Timer) automatisch alle $^1/_{60}$ Sekunden auf. Wir haben allerdings die Möglichkeit zu einem von uns vordefinierten Zeitpunkt einen zusätzlichen Interrupt auszulösen. Zu diesem Zweck müssen wir eine Zeit angeben, die wir als Alarmzeit heranziehen. Die Alarmzeit wird wie die normale Uhrzeit in die Register 8 bis 11 geschrieben, jedoch muß hierfür das Bit 7 in Register 15 gesetzt sein (Rücksetzen nicht vergessen). Stimmen nun die Uhrzeit und die von uns eingestellte Alarmzeit überein, wird ein zusätzlicher IRQ ausgelöst und der Programmzeiger springt zu der



CIA (6526)

Register	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
Register 0 $DC00 (Port A)	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	56320
Register 1 $DC01 (Port B)	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	56321
Register 2 $DC02 (Datenrichtungsregister A)	DPA7	DPA6	DPA5	DPA4	DPA3	DPA2	DPA1	DPA0	56322
Register 3 $DC03 (Datenrichtungsregister B)	DPB7	DPB6	DPB5	DPB4	DPB3	DPB2	DPB1	DPB0	56323
Register 4 $DC04 (Timer A Low-Byte)	TAL7	TAL6	TAL5	TAL4	TAL3	TAL2	TAL1	TAL0	56324
Register 5 $DC05 (Timer A High-Byte)	TAM7	TAM6	TAM5	TAM4	TAM3	TAM2	TAM1	TAM0	56325
Register 6 $DC06 (Timer B Low-Byte)	TBL7	TBL6	TBL5	TBL4	TBL3	TBL2	TBL1	TBL0	56326
Register 7 $DC07 (Timer B High-Byte)	TBM7	TBM6	TBM5	TBM4	TBM3	TBM2	TBM1	TBM0	56327
Register 8 $DC08 (Uhr 1/10 Sekunde)	0	0	0	0	T0	T4	T2	T1	56328
Register 9 $DC09 (Uhr Sekunde)	0	SM4	SM2	SM1	SL0	SL4	SL2	SL1	56329
Register 10 $DC0A (Uhr Minuten)	0	MH4	MH2	MH1	ML0	ML4	ML2	ML1	56330
Register 11 $DC0B (Uhr Stunden)	PM	0	0	MH	ML0	ML4	ML2	ML1	56331
Register 12 $DC0C (Serial Data Register)	S7	S6	S5	S4	S3	S2	S1	S0	56332
Register 13 $DC0D (Interrupt Control Register)	IRQ	0	0	FLG	SP	PLRM	T0	TA	56333
Register 14 $DC0E (Control Register A)	50 HZ	SP MODE	IN MODE	LOAD	RUNMODE	OUTMODE	PB ON	START	56334
Register 15 $DC0F (Control Register B)	ALARM	IN MODE	IN MODE	LOAD	RUNMODE	OUTMODE	PB ON	START	56335

Tabelle 1. Registerbelegung der CIA

in den IRQ-Vektoren angegebenen Adresse. Die Unterscheidung zwischen einem Interrupt durch den Unterlauf eines Timers oder durch Erreichen der Alarmzeit wird durch Register 13 möglich. In diesem Register wird bei Auftreten des IRQ das zuständige Bit (siehe Tabelle 1) gesetzt. Zum besseren Verständnis erwartet Sie in Listing 3 eine kleine Routine, welche bei Erreichen der Alarmzeit die Farbe des Bildschirmrahmens verändert. Diese Routine wird ebenfalls mit SYS 900 gestartet.

Bildschirm mal zwei

Wäre es nicht praktisch, auf dem Monitor zwei voneinander unabhängige Bildschirmseiten unterzubringen? Würde sich ein HiRes-Bild und der dazugehörige Text nicht ungeheuer gut machen? Wenn Ihnen diese oder ähnliche Gedanken schon einmal durch den Kopf gingen, dann sind Sie hier genau richtig. Mittels des VICs (Videocontroller) ist es unserem Computer nicht nur möglich, Zahlen oder Buchstaben auf den Bildschirm zu bringen, er sorgt auch für die HiRes-Grafik, die Sprites und die Verwaltung von 16 KByte Speicherplatz. Da er nebenbei auch noch für die Erzeugung eines normgerechten PAL-Signals zuständig ist, muß er in jedem Augenblick genau wissen, an welcher Stelle sich der Elektronenstrahl Ihres Monitors gerade befindet. Mit einer kleinen Assemblerroutine und ein wenig Bastelei an den Interruptvektoren und -registern unseres Computers können Sie in dem Augenblick, in dem sich der Elektronenstrahl an einer von Ihnen bestimmten Position befindet, zum Beispiel von einem HiRes- auf einen Textbildschirm umschalten (und umgekehrt). Da wir dadurch dem Computer keine Chance lassen, eine Bildschirmseite zu vollenden und einen Bildwechsel vorzunehmen, erscheinen zwei (oder mehrere) Bilder gleichzeitig. Die Trennung dieser Bilder ist abhängig von der Position, an welcher die Umschaltung erfolgt. Die Assemblerroutine in Listing 4 ermöglicht die Darstellung von zwei Textbildschirmen (1024–2023 und 2024–3043) gleichzeitig. In der Speicherstelle 648 können Sie festlegen, auf welchem Bildschirmteil Ihre Eingaben erscheinen sollen. Alle Ausgaben auf den Bildschirm werden ebenfalls in die Bildschirmseite geschrieben, deren Adresse Sie in Speicherstelle 648 angegeben haben. Der Wert, den Sie an diese Adresse POKEn müssen, errechnet sich wie folgt: Adresse des Bildschirmspeichers durch 256. Sobald Sie sich für eine Bildschirmhälfte entschieden haben, können Sie das Programm mit SYS 900 starten (Hier wird der Interruptvektor gesetzt, das eigentliche Programm befindet sich ab Speicherplatz 830. Oberer Bildschirm POKE 648,4, unterer Bildschirm POKE 648,8).

Der VIC hält für uns die Möglichkeit bereit, bei Auftreten eines bestimmten Ereignisses einen Interrupt (IRQ) auszulösen. Zu diesem Zweck stehen die Register 25 und 26 zur Verfügung. Die genaue Belegung dieser Register entnehmen Sie bitte Tabelle 2. Falls mindestens ein Bit aus Register 25 und Register 26 übereinstimmt, wird ein IRQ ausgelöst und der interne Programmzähler springt zu der im IRQ-Vektor (788/789) angegebenen Adresse. Sobald wir diesen Zeiger auf ein eigenes Programm zeigen lassen, wird bei jedem IRQ zuerst unser Programm angesprungen. Das Ende unseres Programmes sollte immer einen unbedingten Sprung in die eigentliche Interruptroutine beinhalten. In unserem Fall müssen wir die Interruptmaske (Register 26) so verändern, daß Register 18 als Auslöser anerkannt wird. Um eine Auslösung an einer bestimmten Stelle des Rasterstrahls zu erreichen, wird in Register 18 (17) die entsprechende Position angegeben. Register 18 hat folglich zwei Aufgaben: 1. wird es gelesen, so gibt es die aktuelle Position des Rasterstrahls an. 2. wird es beschrieben, so gilt der Wert, welcher in das Register geschrieben wurde als die Position des Strahls, an der der IRQ ausgelöst (und Bit 1/Register 25 ist gesetzt) wird. Als letztes Problem bleibt uns noch das CIA. Dieser Chip ist im Normalfall derjenige, welcher die IRQs auslöst. Für unseren Zweck benötigen wir jedoch nur das Interruptsignal, das vom VIC ausgelöst wurde. Aus diesem Grund wird zu Beginn der

```
10 C=56328                                      <226>
20 POKE C+14,PEEK(C+14) OR 128                  <163>
30 POKE C+15,PEEK(C+15) AND 127                 <136>
40 INPUT "ZEIT HHMMSS";A$                       <220>
50 IF LEN(A$)<>6 THEN 40                        <057>
60 H=VAL(LEFT$(A$,2))                           <026>
70 M=VAL(MID$(A$,3,2))                          <239>
80 S=VAL(RIGHT$(A$,2))                          <154>
100 IF H>12 THEN H=H+68                         <001>
110 POKE C+3,16*INT(H/10)+H-INT(H/10)*10        <211>
130 POKE C+2,16*INT(M/10)+M-INT(M/10)*10        <108>
150 POKE C+1,16*INT(S/10)+S-INT(S/10)*10        <090>
160 POKE C,0                                    <224>
180 PRINT"{CLR}"                                <168>
200 C=56328                                     <160>
210 H=PEEK(C+3):M=PEEK(C+2):S=PEEK(C+1):T=
    PEEK(C)                                     <230>
230 F=0:IF H>32 THEN H=H-128:F=1                <013>
240 H=INT(H/16)*10+H-INT(H/16)*16:IF F=0 T
    HEN 280                                     <251>
250 IF H=12 THEN 290                            <126>
260 H=H+12                                      <250>
280 IF H=12 THEN H=0                            <167>
290 M=INT(M/16)*10+M-INT(M/16)*16               <173>
300 S=INT(S/16)*10+S-INT(S/16)*16               <131>
320 T$=RIGHT$("0"+MID$(STR$(H),2,2),2)+":"      <254>
340 T$=T$+RIGHT$("0"+MID$(STR$(M),2,2),2)+
    ":"                                         <023>
360 T$=T$+RIGHT$("0"+MID$(STR$(S),2,2),2)+
    ":"                                         <067>
380 T$=T$+RIGHT$(STR$(T),1)                     <166>
400 PRINT"{HOME,3DOWN}";T$                      <050>
420 GOTO 210                                    <118>
```

Listing 1. Basic-Programm zum Stellen der Uhr

```
NAME : LISTING 2 MC     033E 03A3
-----------------------------------------
033E : A2 03 A0 1E BD 08 DC 85   E8
0346 : FB 29 F0 18 6A 6A 6A 6A   8D
034E : 69 30 EA 99 00 04 A5 FB   6C
0356 : 29 0F 69 30 C8 99 00 04   C9
035E : C8 A9 3A 99 00 04 C8 CA   95
0366 : E0 00 D0 D8 AD 08 DC 69   F7
036E : 2F 99 00 04 4C 31 EA EA   BA
0376 : EA EA EA EA EA EA EA EA   75
037E : EA EA EA EA EA EA 78 A9   30
0386 : 3E 8D 14 03 A9 03 8D 15   03
038E : 03 58 60 EA EA EA EA EA   BA
0396 : EA EA EA E6 FC D0 F4 E6   85
039E : FD A5 FD C9 08 00 00 00   A7
```

Listing 2. Interruptroutine Uhrzeit

```
NAME : LISTING 3 MC     033E 03A3
-----------------------------------------
033E : A2 03 A0 1E BD 08 DC 85   E8
0346 : FB 29 F0 18 6A 6A 6A 6A   8D
034E : 69 30 EA 99 00 04 A5 FB   6C
0356 : 29 0F 69 30 C8 99 00 04   C9
035E : C8 A9 3A 99 00 04 C8 CA   95
0366 : E0 00 D0 D8 AD 08 DC 69   F7
036E : 2F 99 00 04 AD 0D DC C9   35
0376 : 81 F0 03 EE 20 D0 4C 31   2A
037E : EA EA EA EA EA EA 78 A9   30
0386 : 3E 8D 14 03 A9 03 8D 15   03
038E : 03 58 60 EA EA EA EA EA   BA
0396 : EA EA EA E6 FC D0 F4 E6   85
039E : FD A5 FD C9 08 00 00 00   A7
```

Listing 3. Interruptroutine Alarmzeit

```
NAME : LISTING 4 MC     033E 03AD
-----------------------------------------
033E : AD 19 D0 A2 FF 8E 19 D0   7B
0346 : 29 80 C9 80 F0 06 4C 31   05
034E : EA EA EA EA AD 18 D0 C9   38
0356 : 15 F0 07 A9 15 A2 96 18   CB
035E : 90 04 A9 25 A2 00 8D 18   90
0366 : D0 8E 12 D0 4C 81 EA EA   6E
036E : EA EA EA EA EA EA EA EA   6D
0376 : EA EA EA EA EA EA EA EA   75
037E : EA EA EA EA EA EA 78 A9   30
0386 : 3E A2 03 8D 14 03 8E 15   46
038E : 03 A9 F9 8D 1A D0 AD 11   97
0396 : D0 29 7F 8D 11 D0 A9 96   F8
039E : 8D 12 D0 58 60 EA EA EA   52
03A6 : EA EA EA EA EA EA EA 00   CF
```

Listing 4. Bildschirmteilung

Register	Adresse dezimal	Adresse hex	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	53248	$D000	X-Position Sprite Nr. 0							
1	53249	$D001	Y-Position Sprite Nr. 0							
2	53250	$D002	X-Position Sprite Nr. 1							
3	53251	$D003	Y-Position Sprite Nr. 1							
4	53252	$D004	X-Position Sprite Nr. 2							
5	53253	$D005	Y-Position Sprite Nr. 2							
6	53254	$D006	X-Position Sprite Nr. 3							
7	53255	$D007	Y-Position Sprite Nr. 3							
8	53256	$D008	X-Position Sprite Nr. 4							
9	53257	$D009	Y-Position Sprite Nr. 4							
10	53258	$D00A	X-Position Sprite Nr. 5							
11	53259	$D00B	Y-Position Sprite Nr. 5							
12	53260	$D00C	X-Position Sprite Nr. 6							
13	53261	$D00D	Y-Position Sprite Nr. 6							
14	53262	$D00E	X-Position Sprite Nr. 7							
15	53263	$D00F	Y-Position Sprite Nr. 7							
16	53264	$D010	Sprite 7, msb X-Position	Sprite 6, msb X-Position	Sprite 5, msb X-Position	Sprite 4, msb X-Position	Sprite 3, msb X-Position	Sprite 2, msb X-Position	Sprite 1, msb X-Position	Sprite 0, msb X-Position
17	53265	$D011	msb des Rasterregisters (Reg. 18)	Schaltbit für veränderten Hintergrund-farbmodus 1 = eingeschaltet	Schaltbit für Hochauflösungs-modus 1 = eingeschaltet	Schaltbit für Schirm löschen 0 = gelöscht	Schaltbit für Zeilenzahl 0 = 24 Zeilen 1 = 25 Zeilen	Wert der Zeilenverschiebung in Y-Richtung beim Smooth Scrolling		
18	53266	$D012	Rasterregister. Dazu kommt das msb in Bit 7, Register 17							
19	53267	$D013	Lichtgriffel X-Position							
20	53268	$D014	Lichtgriffel Y-Position							
21	53269	$D015	Ein- und Ausschalten von Sprites. 0 = Sprite aus, 1 = Sprite an							
			Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
22	53270	$D016	(unbenutzt)		Reset-Bit, muß 0 sein, damit VIC-II-Chip arbeitet	Schaltbild für Mehrfarbmodus 1 = eingeschaltet	Schaltbit für Spaltenzahl 0 = 38 Spalten 1 = 40 Spalten	Wert der Spaltenverschiebung in X-Richtung beim Smooth Scrolling		
			Sprite-Vergrößerung in Y-Richtung. 0 = normale Größe, 1 = doppelte Größe							
23	53271	$D017	Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
24	53272	$D018	Startadresse Textbildschirm				Startadresse Zeichengenerator oder HiRes-Bitmap			(unbenutzt)
25	53273	$D019	Interrupt-Flag-Register				Lichtgriffel-Interrupt-Flag	Sprite/Sprite-Kollision	Sprite/Hinter-grund-Kollision	Raster-Interrupt-Flag
26	53274	$D01A	Interrupt-Masken-Register				Lichtgriffel-Interrupt	Sprite/Sprite-Kollision	Sprite/Hinter-grund-Kollision	Raster-Interrupt-Maske
			Sprite/Hintergrund-Prioritätenregister. 0 = Sprite hat Priorität, 1 = Hintergrund hat Priorität							
27	53275	$D01B	Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
			Sprite-Mehrfarbmodus-Register. 0 = Normaldarstellung, 1 = Mehrfarbmodus-Darstellung							
28	53276	$D01C	Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
			Sprite-Vergrößerung in X-Richtung. 0 = normale Größe, 1 = doppelte Größe							
29	53277	$D01D	Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
			Sprite/Sprite-Kollision. 0 = keine Berührung. 1 = Berührung							
30	53278	$D01E	Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
			Sprite/Hintergrund-Kollision. 0 = keine Berührung, 1 = Berührung							
31	53279	$D01F	Sprite 7	Sprite 6	Sprite 5	Sprite 4	Sprite 3	Sprite 2	Sprite 1	Sprite 0
32	53280	$D020	(unbenutzt)				Farbe des Bildschirmrahmens			
33	53281	$D021	(unbenutzt)				Hintergrundfarbe Nr. 0 (normale Hintergrundfarbe)			
34	53282	$D022	(unbenutzt)				Hintergrundfarbe Nr. 1			
35	53283	$D023	(unbenutzt)				Hintergrundfarbe Nr. 2			
36	53284	$D024	(unbenutzt)				Hintergrundfarbe Nr. 3			
37	53285	$D025	(unbenutzt)				Sprite-Mehrfarben-Register Nr. 0			
38	53286	$D026	(unbenutzt)				Sprite-Mehrfarben-Register Nr. 1			
39	53287	$D027	(unbenutzt)				Sprite 0, Farbe			
40	53288	$D028	(unbenutzt)				Sprite 1, Farbe			
41	53289	$D029	(unbenutzt)				Sprite 2, Farbe			
42	53290	$D02A	(unbenutzt)				Sprite 3, Farbe			
43	53291	$D02B	(unbenutzt)				Sprite 4, Farbe			
44	53292	$D02C	(unbenutzt)				Sprite 5, Farbe			
45	53293	$D02D	(unbenutzt)				Sprite 6, Farbe			
46	53294	$D02E	(unbenutzt)				Sprite 7, Farbe			

Tabelle 2. Alle Register des Video-Chips auf einen Blick

Routine mittels des IRQ-Flags in Register 25 abgefragt, ob der VIC den IRQ angefordert hat.

Nachträglich Befehle einfügen

Basic-Zeilen in ein laufendes Programm einfügen? Wollen Sie Funktionen austesten oder ein Basic-Programm entwickeln, welches sich selbst verändert? Mit unserem Basic »Listing 5« und der Assemblerroutine aus »Listing 5 MC« können Sie ohne Schwierigkeiten alle beliebigen Daten in Ihr Basicprogramm aufnehmen, ohne das Programm stoppen zu müssen. Für diese Aufgabe stellt das Betriebssystem unseres Computers einige sehr effektive Routinen zur Verfügung. Unsere Aufgabe besteht nur noch darin, die einzufügenden Daten in den Basic-Eingabepuffer (512–600) zu schreiben und die Routine aufzurufen. Nach Aufruf des Programms wird der Vektor zur Eingabe einer Zeile auf eine Adresse innerhalb unseres Programms gesetzt. Daraufhin wird die komplette Betriebssystemroutine zur Einfügung von Programmzeilen abgearbeitet. Zum Ende dieser Routine würde der Programmzeiger in die Eingabe-Warteschleife springen. Durch die anfängliche Änderung des entsprechenden Vektors wird der Programmzeiger jedoch wieder auf unsere Routine umgelenkt. Hier wird nun der Vektor wieder auf den ursprünglichen Wert verändert und das Basic-Programm erneut gestartet. Wie bereits zu Anfang erwähnt, eignet sich dieser kleine Trick hervorragend zur Eingabe von Funktionen.

Basic optimieren

Sie haben sich sicher schon oft über die schwache Leistung unseres Basic-Interpreters geärgert. Je komplexer die Anforderungen an ein Basic-Programm sind, desto langweiliger wird die Geschwindigkeit der betreffenden Programme. Viele scheuen jedoch den Schritt in Richtung Maschinensprache und ärgern sich weiterhin mit ihren »Schlafmützen«-Programmen. Betrachtet man aber die Arbeitsweise des Interpreters ein wenig genauer, so bieten sich einige Möglichkeiten zur Beschleunigung von Basic-Programmen. Einmal abgesehen von den verwendeten Algorithmen und der Struktur des Programmes gibt es mehrere Tricks, um den Interpreter ein wenig anzutreiben.

Variable

Sobald in einem Basic-Programm eine Variable benötigt wird, beginnt der Interpreter, vom Variablenstart an aufwärts zählend, nach der Variable zu suchen. Je später eine Variable definiert wurde, desto länger dauert der Suchvorgang. Ähnlich verhält es sich bei Konstanten. Werden Konstante im Programm angegeben, so müssen diese bei jedem Durchlauf erneut umgewandelt werden. Definiert man Konstante als Variable, dann fällt die Umrechnung nur einmal an.

(1) Konstante, Variable, und Felder in der Reihenfolge ihrer Zugriffshäufigkeit vordefinieren. Möglichst ein- oder zweistellige Variablennamen verwenden.

Unterprogramme

Unterprogramme müssen vom Interpreter, wie Variablen, erst einmal gesucht werden. Da die Suche mit der niedrigsten Zeilennummer beginnt, findet er Zeilen am Anfang natürlich schneller.

(2) Unterprogramme sollten am Anfang eines Programmes stehen.

Verzweigungen

IF-THEN-Abfragen mit mehreren, durch AND-Verknüpfungen verbundenen, Bedingungen sollten durch Hintereinanderlegen von mehreren IF-THEN-Abfragen verschachtelt werden. Beispiel:

a. IF (A kleiner 100 AND B größer 0) THEN
b. IF A kleiner 100 THEN IF B größer 0 THEN

Version b bricht sofort nach Nichterfüllung der ersten Bedingung ab und ist somit schneller.

(3) IF-THEN-Abfragen mit mehreren Bedingungen verschachteln.

Leerzeichen

REMs und Leerstellen verzögern den Programmlauf, da der Interpreter sie ja ignorieren muß. Je mehr Befehle sich in einer Programmzeile befinden, desto seltener muß der Computer nach neuen Zeilenanfängen suchen.

(4) Programme kompakter verfassen.

Strings

Die Garbage-Collection-Routine des Betriebssystems beseitigt den »String-Müll«. Werden mehrere Strings definiert, fällt auch mehr »Müll« an. Das Tragische an dieser Routine ist die Geschwindigkeit: verdoppelt sich die Anzahl der Strings, vervierfacht (!) sich die Laufzeit. Aus diesem Grund bringt das Anlegen von String-Konstanten auch keinen Zeitgewinn.

(5) So wenig Strings wie möglich benutzen.

Potenzieren

Falls in Ihrem Programm eine ganzzahlige Potenzierung (x^2, x^4..) vorkommt, ist es ratsam, die Potenzierung durch eine Mehrfachmultiplikation zu ersetzen. Da die Potenzierungsroutine des Interpreters auch Potenzierungen mit Brüchen berrechnen kann, ist sie $1^1/_2$ mal langsamer als die entsprechende Multiplikation.

(6) Potenzierung durch Multiplikation ersetzen.

Interrupt

Das CIA 1 löst alle $^1/_{60}$ Sekunden einen IRQ aus, um daraufhin den Programmzeiger in die Interruptroutine springen zu lassen. Hier wird die Tastatur abgefragt, die Zeit erhöht und noch einige andere Dinge erledigt. Die Bearbeitung und Beachtung des Interrupts zweigt natürlich einen Teil der ach so kostbaren Rechenzeit ab. Wenn wir den Interrupt verhindern, solange wir die Tastatur nicht benötigen, so kann auch mit dieser Technik ein Basic-Programm beschleunigt werden. POKE 56333,31 verhindert den IRQ, POKE 56333,159 gibt den IRQ wieder frei.

(7) Sperren des Interrupts.

Probieren Sie es aus. Verändern Sie Ihre alten Programme in der beschriebenen Weise und vergleichen Sie die Geschwindigkeit. Sie werden überrascht sein.

Zeitsparen einmal anders

Basic-Programme werden meist erst durch den Einsatz von Assemblerroutinen zu Programmen mit vernünftiger Laufgeschwindigkeit. Der Sprung in diese Routinen wird häufig durch einen SYS-Befehl realisiert. Die Parameterübergabe wird in diesen Fällen meist durch vorangehende POKE-Befehle bewerkstelligt. Sie sollen sich nun nicht mehr länger mit dieser umständlichen, uneleganten und vor allem langsamen Methode begnügen müssen. In unserem Computer verbirgt sich eine Funktion, die von vielen Handbüchern schlicht vergessen oder nur im Vorübergehen behandelt wird. Es handelt sich hierbei um die »USR-Funktion«. Im Gegensatz zum SYS-Befehl beinhaltet die USR-Funktion bereits eine vollständige Parameterübergabe. Sie kann also genauso benutzt werden wie alle anderen Funktionen des Basic-Interpreters (CHR$(x), ASC(x) ...). Der Vorteil dieser Funktion ist die freie Definition der eigentlichen Operation durch den Programmierer. Aber zunächst einmal die Form der Funktion: x1=USR(x2), wobei x2 der Wert ist, mit welchem die Routine arbeitet und x1 das Ergebnis der Operation darstellt. Die Parameter x1 und x2 können jede beliebige Form annehmen, wie zum Beispiel: Zahl, Variable, Zeichen, String (natürlich muß der Parameter zur jeweiligen Operation passen). x2 wird beim Aufruf der Routine automatisch in den FAC (Fließkommaakkumulator) übernommen. Analog dazu wird der Wert des FAC am Ende des Programmlaufs in x1 geschoben. Die weitere Funktionsweise der USR-Funktion ist ähnlich dem

```
0 REM **** STARTZEILE ****                         <099>
1000 INPUT A$:IF A$=""THEN END                     <197>
1010 FOR AA=0 TO LEN(A$)-1:POKE 513+AA,ASC
     (MID$(A$,AA+1,1)):NEXT:POKE 513+AA,0          <071>
1020 POKE 11,AA:SYS 830                            <100>
```

Listing 5a. Basic-Teil einfügen

```
NAME : LISTING 5 MC     033E 037B
----------------------------------
033E : A9 55 8D 02 03 A9 03 8D  DA
0346 : 03 03 A9 00 85 7A A9 02  0C
034E : 85 7B A9 30 4C 9C A4 A9  91
0356 : 83 8D 02 03 A9 A4 8D 03  7D
035E : 03 4C 71 A8 EA EA EA EA  80
0366 : EA EA EA EA EA EA EA EA  65
036E : EA EA EA EA EA EA EA EA  6D
0376 : EA EA EA EA EA EA EA EA  75
```

Listing 5b. Maschinen-Teil einfügen

```
NAME : LISTING 6 MC     033E 0353
----------------------------------
033E : 20 6B E2 20 E2 BA 60 EA  2C
0346 : EA EA EA EA EA EA EA EA  45
034E : EA EA EA EA EA 9C A4 A9  3F
```

Listing 6. USR-Routine

```
NAME : LISTING 7 MC     033C 03F3
----------------------------------
033C : 20 8D AD A6 2F, A5 30 86  51
0344 : 5F 85 60 C5 32 D0 04 E4  BA
034C : 31 F0 1D A0 00 B1 5F C8  ED
0354 : C5 45 D0 06 A5 46 D1 5F  43
035C : F0 17 C8 B1 5F 18 65 5F  4B
0364 : AA C8 B1 5F 65 60 90 D7  16
036C : A2 E2 86 22 A9 03 4C 45  D4
0374 : A4 C8 B1 5F 18 65 5F 85  0A
037C : 24 C8 B1 5F 65 60 85 25  16
0384 : C8 B1 5F 20 96 B1 85 5F  CC
038C : 84 60 24 0E 30 1F 20 A2  CD
0394 : BB 18 90 04 20 67 B8 18  50
039C : A5 5F 69 05 85 5F 90 02  85
03A4 : E6 60 A4 60 C5 24 90 EC  89
03AC : C4 25 90 E8 60 20 D5 03  A8
03B4 : 20 0C BC 18 A5 5F 69 02  0B
03BC : 85 5F 90 02 E6 60 C5 24  26
03C4 : 90 06 A5 60 C5 25 B0 E4  DF
03CC : 20 D5 03 20 6F B8 4C B4  F3
03D4 : 03 A0 00 B1 5F AA C8 B1  2F
03DC : 5F A8 8A 4C 91 B3 41 52  1C
03E4 : 52 41 59 20 4E 4F 54 20  22
03EC : 46 4F 55 4E C4 00 00 00  45
```

Listing 7. Array-Berechnungen

```
NAME : LISTING 8 MC     033E 035D
----------------------------------
033E : 20 FD AE 20 9E B7 8A 48  6F
0346 : 20 FD AE 20 9E B7 68 A8  AF
034E : 18 20 F0 FF 20 FD AE 4C  F8
0356 : A4 AA EA EA EA EA EA 17  47
```

Listing 8. Print AT-Simulation

SYS-Befehl beziehungsweise dem Aufruf einer normalen Basic-Funktion. Nach Aufruf der Funktion wird ein Assemblerprogramm angesprungen. Die Adresse dieses Programmes muß jedoch nicht bei jedem Aufruf erneut angegeben werden, sondern wird im sogenannten USR-Vektor (785/786) vordefiniert. Falls Sie die USR-Funktion bereits einmal ausprobieren wollten und als Antwort ein »SYNTAX ERROR« erhielten, so lassen Sie sich bitte davon nicht irritieren. Der USR-Vektor wird, wie alle anderen Vektoren, während des »Start-Resets« auf eine festgelegte Routine des Betriebssystems gelegt. Der Startwert des USR-Vektors zeigt dann auf eine Routine, die einen »SYNTAX ERROR« ausgibt. Aber nun zu unserer ersten Anwendung. Sie finden in Listing 6 eine Routine, die von dem angegebenen Wert den Sinuswert errechnet und diesen mit 10 multipliziert (Start an Adresse 830). Innerhalb von 100 Durchläufen erreicht unsere Routine einen Vorsprung von $^{18}/_{60}$ Millisekunden gegenüber einem entsprechenden Basic-Programm. Wie Sie an diesem Beispiel erkennen können, ermöglicht es die USR-Funktion, sowohl von Basic aus zugängliche Routinen wie auch Interpreter-interne Operationen aufzurufen und für eigene Zwecke zu nutzen. In Listing 7 haben wir für Sie bereits eine etwas anspruchsvollere Routine vobereitet. Dieses Programm berechnet aus einem beliebigen Variablenfeld (Array) die Summe der einzelnen Argumente. In diesem Fall muß bei Aufruf der USR-Funktion der Namen des Feldes angegeben werden und der USR-Vektor auf Adresse 828 gerichtet sein. Wenn Sie dieses Vorhaben von Basic aus realisieren möchten, müssen Sie schon eine Weile warten. Sie werden bereits bemerkt haben, daß Ihnen zur sinnvollen Nutzung der USR-Funktion nicht nur der Befehlssatz der 65xx-Prozessoren geläufig sein sollte. Der richtige Einsatz der ROM-Routinen ist mindestens genauso wichtig. Zu diesem Zweck ist die Benutzung eines ROM-Listings unumgänglich. Nachfolgend erwartet Sie eine kurze Aufstellung der wichtigsten Routinen. Bitte achten Sie bei der Benutzung der Arithmetik-Sequenzen auf die Einsprungadressen. Teilweise befindet sich zu Beginn der Routinen eine Abfrage auf FAC=0. Diese Abfrage sollte beim Aufruf über den USR-Vektor übergangen werden (Weitere ausführliche Informationen über Rechenoperationen und deren Anwendung finden Sie in dem Artikel »Rechnen in Maschinensprache«).

B849 : FAC = FAC+0.5
B850 : FAC = Konstante−FAC
B853 : FAC = ARG−FAC
B867 : FAC = FAC+ARG
BA28 : FAC = Konstante*FAC
BA2B : FAC = ARG*FAC
BA8C : ARG = Konstante
BAE2 : FAC = FAC*10
BB0F : FAC = Konstante/FAC
BB12 : FAC = ARG/FAC
BBA2 : FAC = Konstante
BBFC : FAC = ARG
BC0C : ARG = FAC
BF78 : FAC = $ARG^{Konstante}$
BF7B : FAC = ARG^{FAC}

Wie bereits erwähnt, können natürlich alle von Basic aus nutzbaren Routinen ebenfalls verwendet werden (SQR, SIN ...). Abschließend noch eine Anregung. Versuchen Sie einmal, mehrere Basic-Befehle unter der USR-Funktion zusammenzufassen. Auch durch diesen Trick kann man Basic beschleunigen.

SYS und mehr

Gerade haben wir behauptet, die USR-Funktion sei der Weisheit letzter Schluß, und jetzt beschäftigen wir uns doch wieder mit dem SYS-Befehl. Bitte glauben Sie nicht, wir wären inkonsequent. Es soll hier nur noch auf eine weitere Technik der Datenübergabe hingewiesen werden, die übrigens auch im Umgang mit der USR-Funktion verwendet werden kann. Auf einfachste Weise ist es möglich, nicht nur einen Parameter (USR), sondern beliebig viele zu übernehmen. Hierfür bieten sich gleich drei Routinen an:

AE83 : holt das nächste Element eines Ausdrucks in FAC
B79E : holt ein Byte in das X-Register
0073 : holt nächstes Zeichen in Akku

Das Betriebssystem hält sogar noch einige Routinen zur Prüfung der zu übergebenden Parameter bereit. Aber bitte bemühen Sie in diesen Fällen Ihr ROM-Listing. Zum Abschluß finden Sie in Listing 8 eine PRINT-AT-Simulation. Der Einsprung erfolgt über SYS 830, spalte, zeile, »text« (siehe auch »Rechnen in Maschinensprache«). (Erhart/do)

Tips & Tricks kunterbunt

Auf den nächsten Seiten finden Sie eine Auswahl an hervorragenden Programmen, Unterroutinen und Programmiertips für fast jeden Gebrauch. Erwähnenswert sind vor allem ein außergewöhnliches Turbo-Tape und ein neuartiges Prinzip der Dateiverwaltung, das sich auch hervorragend als Lernprogramm für angehende Taxi-Fahrer eignet.

Unsere beliebte Tips & Tricks-Rubrik aus dem 64'er-Stammheft hat einen nicht unerheblichen Nachteil: Viele nützliche und sehr gute Programme sind einerseits zu kurz für einen eigenständigen Artikel, aber andererseits auch zu lang für eine Veröffentlichung unter »Tips & Tricks«. Im Laufe der Zeit hat sich da natürlich eine ganze Menge an Programmen angesammelt. In diesem Sonderheft haben wir nun die Gelegenheit, diesen Stapel auszubauen und Ihnen eine geballte Ladung von Tips & Tricks-Listings zu präsentieren.

Übrigens: Wenn Sie noch kein routinierter 64'er-Leser sind, sollten Sie vor dem Abtippen unbedingt unsere Eingabehinweise zum Checksummer und dem MSE auf den Seiten 119 und 120 lesen.

Der (fast) perfekte Listschutz

Bei diesem Programm »Invisible« (Listing 1) handelt es sich um eine Hilfe zur Listschutzerstellung. Nach Behandlung eines Basic-Programms mit »Invisible« sind davon nur noch die Zeilennummern sichtbar. Dies wird durch fünf hinter die Zeilennummer eingefügte Doppelpunkte erreicht, von denen der erste durch ein Null-Byte ersetzt wird.

Das ist soweit nichts Besonderes, doch mußte man bei ähnlichen Arten des Listschutzes bisher die Doppelpunkte von Hand einfügen. »Invisible« nimmt Ihnen diese Arbeit ab. Es fügt auf Tastendruck pro Zeile fünf Doppelpunkte ein. Zeilen, in die keine fünf Zeichen mehr passen, werden nicht verändert.

Ebenfalls auf Tastendruck werden die »Doppelpunktzeilen« unsichtbar gemacht. Hier bleiben Zeilen mit einer falschen Zahl an Doppelpunkten unberücksichtigt.

Will man sein Programm wieder sehen, sei es, um es zu ändern oder nur aus Spaß an der Freude, so kann man mit »Invisible« sämtliche Zeilen, die wie oben beschrieben geschützt wurden, wieder sichtbar machen. Sogar die fünf Doppelpunkte werden wieder entfernt. Hier bleiben nun Zeilen mit weniger als fünf Doppelpunkten unverändert (sie waren zuvor ja auch nicht geschützt).

Beschreibung der einzelnen Programmteile:

1. Doppelpunkte einfügen:

Als erstes wird die Startadresse der ersten Zeile in die Variable »by« geschrieben. Dann wird deren Zeilennummer in »lb« und »hb« festgestellt. Es folgt dann noch die Adresse der nächsten Zeile. Die Werte dieser Variablen müssen in Speicherzellen mit POKE zwischengespeichert werden, da sie bei der Änderung der Zeilen während des Einfügens der Doppelpunkte gelöscht werden würden.

Nun wird in der Variablen »bn« die Adresse der nächsten Zeile aus »12« und »h2« berechnet. Diese wird zur Bestimmung der Zeilenlänge benötigt. Es folgt noch eine Abfrage auf eventuell bereits vorhandene Doppelpunkte.

Steht in der Zeile kein Doppelpunkt, so wird der Bildschirm gelöscht und ein »LIST aktuelle Zeile« sowie ein »GOTO....« links oben auf den Bildschirm geschrieben. Dies wird für den nachfolgenden Direktmodus benötigt. Es wird nun anhand der Länge der Zeilennummer und der restlichen Zeile berechnet, wie viele Zeilen man mit Hilfe des Tastaturpuffers nach unten gehen muß, um auf das »GOTO...« zu kommen.

Nun wird noch die Länge der Zeilennummer festgestellt, da sie zum Einfügen der Doppelpunkte übersprungen werden muß. Dies geschieht in den Zeilen 63032 und folgende.....

Jetzt wird die zu bearbeitende Zeile gelistet. In Zeile 63240 wird mit Hilfe von Steuerzeichen alles, was hinter einer Zeilennummer steht, um fünf Zeichen nach rechts verschoben, wobei dann die Doppelpunkte eingefügt werden.

2. Doppelpunkte löschen:

Dieser Programmteil läuft ziemlich genauso ab wie der Teil »Doppelpunkte einfügen«. Lediglich die Abfrage der Doppelpunkte und deren »Einfügen« ist anders: Es wird nicht abgefragt, ob schon ein Doppelpunkt vorhanden ist, sondern ob es deren genau fünf sind. Statt dem Einfügen der Doppelpunkte werden diese mit fünf Leerzeichen überschrieben.

3. Doppelpunktzeichen unsichtbar machen:

Das Bestimmen der Zeilendaten geht analog zu den vorigen Teilen. Dann werden die ersten sechs Byte aus der eigentlichen Basic-Zeile gelesen und auf Doppelpunkte überprüft. Sind die ersten fünf Byte Doppelpunkte, so wird in das erste dieser fünf Byte eine Null gePOKEt. Dadurch listet der Computer nur noch die Zeilennummern.

4. Zeilen wieder sichtbar machen:

Im Unterschied zur vorherigen Routine wird hier geprüft, ob das erste Byte (in »a« gespeichert) eine Null und die restlichen vier Byte Doppelpunkte (Code 58) sind. Es stehen also wieder fünf Doppelpunkte hinter der Zeilennummmer und das Programm läßt sich wieder vollständig LISTen.

5. Löschroutine:

Hier wird die vorgegebene Zeilennummer (63000) in die linke obere Ecke des Bildschirms geschrieben und mittels des bereits oben beschriebenen »programmierten Direktmodus« in der ersten Zeile des Bildschirms (in der die Zeilenzahl steht), ein »carriage return« ausgeführt. Dadurch wird die Zeile gelöscht.

So arbeitet man mit »INVISIBLE«

1. Laden des zu bearbeitenden Programms.
2. Eingeben: POKE 43, PEEK(45)-2:POKE 44,PEEK(46)

In seltenen Fällen kann es passieren, daß der C 64 einen »illegal quantity error« meldet (Ihr Programm hat dann eine ungünstige Endadresse). Dies läßt sich beheben, indem Sie noch irgendwo eine Zeile einfügen (zum Beispiel ein REM), und die POKEs noch einmal eingeben.

3. Laden von »Invisible« (Listing 1)
4. Eingeben: POKE 43,1 : POKE 44,8
5. Starten von »Invisible« mit »RUN 63000«.

(Volker Imre/tr)

Besondere GOSUBs

Da das C 64-Basic nicht über Label verfügt, habe ich ein Programm geschrieben, mit dem in Unterprogramme mit beliebigem Namen gesprungen werden kann. Im Programmlisting erscheint an Stelle des GOSUB-Befehls nur der Name des entsprechenden Unterprogramms. Dadurch wird die Lesbar-

keit des Programms wesentlich verbessert und die Unterprogramme können frei verschoben werden, ohne Einsprungadressen verändern zu müssen. Mit einer geeigneten MERGE-Routine kann man nun aus einer Unterprogrammsammlung bequem Programme zusammenstellen. Die Benutzung sieht wie folgt aus: Auf ein vorangestelltes Ausrufezeichen folgt der Name des aufzurufenden Unterprogramms gefolgt von einem abschließenden Doppelpunkt. Alle Zeichen zwischen dem Ausrufezeichen und dem Doppelpunkt werden als Name interpretiert. Falls Basic-Befehlsworte im Namen vorkommen, muß der Name in Anführungszeichen gesetzt werden.

Damit das Unterprogramm gefunden werden kann, steht am Anfang des Unterprogramms eine REM-Zeile, gefolgt vom Namen des Unterprogramms. Dabei ist zu beachten, daß bei Basic-Befehlsworten die Anführungszeichen auch in der REM-Zeile stehen müssen. Listing 2 zeigt die eigentliche Befehlserweiterung; Listing 3 ein kleines Anwendungsbeispiel. (Norbert Reuber/tr)

```
63000 REM MENUE                                    <193>
63001 PRINT CHR$(147)CHR$(5)CHR$(14):POKE
      53281,0:POKE 53280,0                         <153>
63002 PRINT"{4DOWN}"TAB(2)"-- 1 --{2SPACE}
      ==> DOPPELPUNKTE EINFUEGEN."                 <052>
63003 PRINT"{DOWN}"TAB(2)"-- 2 --{2SPACE}=
      => DOPPELPUNKTE ENTFERNEN."                  <173>
63004 PRINT"{DOWN}"TAB(2)"-- 3 --{2SPACE}=
      => ZEILEN UNSICHTBAR MACHEN"                 <250>
63005 PRINT"{DOWN}"TAB(2)"-- 4 --{2SPACE}=
      => ZEILEN SICHTBAR MACHEN."                  <038>
63006 PRINT"{DOWN}"TAB(2)"-- Q --{2SPACE}=
      => ENDE !"                                   <140>
63007 A$="":GET A$:IF A$=""THEN 63007              <161>
63008 IF A$="Q"THEN GOTO 63115                     <184>
63009 Z=VAL(A$):IF Z<1 OR Z>4 THEN RUN 630
      01                                           <079>
63010 IF Z=3 THEN UN=1:GOSUB 63055                 <156>
63011 IF Z=4 THEN UN=0:GOSUB 63055                 <185>
63012 IF Z=1 THEN GOSUB 63016                      <089>
63013 IF Z=2 THEN GOSUB 63074                      <138>
63014 UN=2:GOTO 63001                              <141>
63015 REM DOPPELPUNKTE EINFUEGEN                   <104>
63016 BY=2049:REM STARTADRESSE 1. ZEILE            <160>
63017 LB=PEEK(2051):HB=PEEK(2052):REM ZEIL
      ENNUMMER 1. ZEILE                            <225>
63018 L2=PEEK(2049):H2=PEEK(2050):REM ADRE
      SSE NAECHSTE ZEILE                           <059>
63019 POKE 49160,INT(BY/256):POKE 49159,BY
      -INT(BY/256)*256                             <059>
63020 POKE 49153,LB:POKE 49154,HB                  <237>
63021 POKE 49155,L2:POKE 49156,H2                  <177>
63022 ZE=LB+256*HB:REM ZEILENNUMMER AUS LO
      W- UND HIGHBYTE                              <096>
63023 IF ZE>62999 THEN 63051:REM TEST OB S
      CHUTZPROGRAMM ERREICHT                       <082>
63024 BN=L2+256*H2:REM ADRESSE NAECHSTE ZE
      ILE                                          <001>
63025 IF BN-BY>60 THEN PRINT"{CLR,8DOWN}ZE
      ILE"ZE;"IST ZU LANG"                         <207>
63026 IF BN-BY>60 THEN PRINT"TASTE !":POKE
       198,0:WAIT 198,1:POKE 198,0:BY=BN:G
      OTO 63045                                    <096>
63027 IF PEEK(BY+4)=58 THEN PRINT"{CLR,8DO
      WN}IN";ZE;"IST BEREITS EIN DOPPELPUN
      KT !"                                        <200>
63028 IF PEEK(BY+4)=58 THEN PRINT"TASTE !"
      :POKE 198,0:WAIT 198,1:POKE 198,0            <245>
63029 IF PEEK(BY+4)=58 THEN BY=BN:GOTO 630
      45                                           <167>
63030 PRINT"{CLR}LIST";LB+256*HB:PRINT"{HO
      ME,5DOWN}GO63032":":PRINT"GO63032":PR
      INT"GO63032"                                 <137>
63031 GOSUB 63111:STOP:REM CURSOR DOWN BES
      TIMMEN                                       <169>
63032 IF ZE>9999 THEN X=6:GOTO 63038               <095>
63033 IF ZE>999 THEN X=5:GOTO 63038                <079>
63034 IF ZE>99 THEN X=4:GOTO 63038                 <205>
63035 IF ZE>9 THEN X=3:GOTO 63038                  <018>
63036 IF ZE<=9 THEN X=2:GOTO 63038                 <093>
63037 REM ↑ FESTSTELLEN WIEVIEL NACH RECHT
      S WEGEN ZEILENNUMMER ↑                       <021>
63038 PRINT"{HOME,2DOWN}";                         <133>
63039 FOR I=1 TO X:PRINT"{RIGHT}";:NEXT            <087>
63040 PRINT"{5INST}:::::":REM EINFUEGEN DO
      PPELPUNKTE                                   <092>
63041 PRINT"{4DOWN}GOTO 63044:{5UP}"               <125>
63042 POKE 631,145:POKE 632,145:POKE 633,1
      45:POKE 634,145:POKE 635,13                  <031>
63043 POKE 636,17:POKE 637,17:POKE 638,17:
      POKE 639,17:POKE 640,13:POKE 198,10:
      STOP                                         <246>
63044 BY=PEEK(49155)+256*PEEK(49156)+5:REM
       ADRESSE ALTE ZEILE AUS LB UND HB            <083>
63045 POKE 49159,BY-INT(BY/256)*256:POKE 4
      9160,INT(BY/256)                             <012>
63046 L2=PEEK(BY):H2=PEEK(BY+1):REM ADRESS
      E NEUE ZEILE LOW UND HIGHBYTE                <020>
63047 POKE 49155,L2:POKE 49156,H2                  <203>
63048 LB=PEEK(BY+2):HB=PEEK(BY+3):REM ZEIL
      ENNUMMER NEUE ZEILE LOW- UND HIGHBYT
      E                                            <052>
63049 ZE=LB+256*HB:REM ZEILENNUMMER NEUE Z
      EILE                                         <241>
63050 IF ZE<63000 THEN GOTO 63024:REM SCHU
      TZPROGRAMM ERREICHT ?                        <114>
63051 PRINT"{CLR,10DOWN}ENDZEILE ERREICHT
      !"                                           <000>
63052 FOR I=1 TO 1000:NEXT:GOTO 63001              <132>
63053 REM ZEILEN UNSICHTBAR MACHEN                 <116>
63054 REM ZEILEN SICHTBAR MACHEN                   <184>
63055 PRINT"{2DOWN,2RIGHT}EINEN MOMENT !":
      BY=2049                                      <117>
63056 LB=PEEK(2049):HB=PEEK(2050)                  <126>
63057 ZE=PEEK(2051)+256*PEEK(2052)                 <161>
63058 IF ZE>62999 THEN RETURN                      <035>
63059 A=PEEK(BY+4)                                 <253>
63060 B=PEEK(BY+5)                                 <004>
63061 C=PEEK(BY+6)                                 <011>
63062 D=PEEK(BY+7)                                 <018>
63063 E=PEEK(BY+8)                                 <025>
63064 F=PEEK(BY+9):REM ↑ ERSTE 6 BYTE AUS
      DER EIGENTLICHEN ZEILE LESEN ↑               <206>
63065 IF UN=1 THEN IF A=58 AND B=58 AND C=
      58 AND D=58 AND E=58 AND F<>58 THEN
      POKE BY+4,0                                  <073>
63066 IF UN=0 THEN IF A=0 AND B=58 AND C=5
      8 AND D=58 AND E=58 THEN POKE BY+4,5
      8                                            <250>
63067 REM ↑ TEST AUF 5 DOPPELPUNKTE BZW AU
      F 1. BYTE = 0 ↑                              <001>
63068 BY=LB+256*HB:REM NEUE ADRESSE                <230>
63069 ZE=PEEK(BY+2)+256*PEEK(BY+3):REM NEU
      E ZEILENNUMMER                               <152>
63070 IF ZE>62999 THEN RETURN                      <047>
63071 LB=PEEK(BY):HB=PEEK(BY+1):REM LOW- U
      ND HIGHBYTE ADRESSE NAECHSTE ZEILE           <221>
63072 GOTO 63059                                   <070>
63073 REM DOPPELPUNKTE ENTFERNEN                   <229>
63074 PRINT"{2DOWN}EINENMOMENT BITTE !":BY
      =2049:REM ADRESSE ERSTE ZEILE                <168>
63075 LB=PEEK(2051):HB=PEEK(2052):REM LB U
      ND HB DER NUMMER DER NAECHSTEN ZEILE         <167>
63076 L2=PEEK(2049):H2=PEEK(2050):REM LB U
      ND HB DER ADRESSE DER NAECHSTEN ZEIL
      E                                            <081>
63077 POKE 49160,INT(BY/256):POKE 49159,BY
      -INT(BY/256)*256                             <117>
63078 POKE 49153,LB:POKE 49154,HB                  <039>
63079 POKE 49155,L2:POKE 49156,H2                  <235>
63080 ZE=LB+256*HB:REM BERECHNEN DER ZEILE
      NNUMMER                                      <041>
63081 IF ZE>62999 THEN 63108:REM TEST OB I
      M SCHUTZPROGRAMM                             <162>
63082 BN=L2+256*H2:REM ADRESSE DER NAECHST
      EN ZEILE FUER LAENGENBESTIMMUNG              <027>
63083 FOR I=4 TO 8:IF PEEK(BY+I)<>58 THEN
      PRINT"{CLR,8DOWN}IN"ZE"SIND KEINE 5
      DOPPEL";                                     <060>
63084 IF PEEK(BY+I)<>58 THEN PRINT"PUNKTE"
      :PRINT"TASTE"                                <050>
63085 IF PEEK(BY+I)<>58 THEN POKE 198,0:WA
      IT 198,1:POKE 198,0                          <048>
63086 IF PEEK(BY+I)<>58 THEN BY=BN:GOTO 63
      102                                          <250>
```

Listing 1. »Invisible«, der (fast) perfekte Listschutz

```
63087 NEXT I                                      <176>
63088 PRINT"{CLR}LIST";LB+256*HB                  <216>
63089 PRINT"{HOME,5DOWN}GO63090:":PRINT"GO
      63090":PRINT"GO63090":GOSUB 63111:ST
      OP                                          <223>
63090 IF ZE>9999 THEN X=6:GOTO 63095              <153>
63091 IF ZE>999 THEN X=5:GOTO 63095               <137>
63092 IF ZE>99 THEN X=4:GOTO 63095                <007>
63093 IF ZE>9 THEN X=3:GOTO 63095                 <076>
63094 IF ZE<=9 THEN X=2:GOTO 63095:REM ↑ B
      ESTIMMEN DER ZAHL CURSOR RECHTS WEGE
      N ZN                                        <046>
63095 PRINT"{HOME,2DOWN}";                        <190>
63096 FOR I=1 TO X:PRINT"{RIGHT}";:NEXT           <144>
63097 PRINT"{5SPACE}":REM DOPPELPUNKTE ENT
      FERNEN                                      <055>
63098 PRINT"{4DOWN}GOTO 63101:{5UP}"              <241>
63099 POKE 631,145:POKE 632,145:POKE 633,1
      45:POKE 634,145:POKE 635,13                 <088>
63100 POKE 636,17:POKE 637,17:POKE 638,17:
      POKE 639,17:POKE 640,13:POKE 198,10:
      STOP                                        <047>
63101 BY=PEEK(49155)+256*PEEK(49156)-5:REM
      ADRESSE NEUE ZEILE                          <184>
63102 POKE 49159,BY-INT(BY/256)*256:POKE 4
      9160,INT(BY/256)                            <069>
63103 L2=PEEK(BY):H2=PEEK(BY+1):REM ADRESS
      E NAECHSTE ZEILE                            <185>
63104 POKE 49155,L2:POKE 49156,H2                 <004>
63105 LB=PEEK(BY+2):HB=PEEK(BY+3):REM NUMM
      ER AKTUELLE ZEILE                           <024>
63106 ZE=LB+256*HB:REM NUMMER BERECHNEN           <070>
63107 IF ZE<63000 THEN GOTO 63082                 <203>
63108 PRINT"{CLR,10DOWN}ENDZEILE ERREICHT
      !"                                          <057>
63109 FOR I=1 TO 1000:NEXT                        <029>
63110 GOTO 63001                                  <039>
63111 POKE 631,19:POKE 632,13:POKE 633,17         <128>
63112 POKE 634,13:POKE 198,4:STOP                 <136>
63113 REM ↑ CURSOR MITTELS TASTATURPUFFER
      AUF GOTO SETZEN ↑                           <136>
63114 REM LOESCHROUTINE                           <158>
63115 ZE=63000                                    <112>
63116 POKE 49165,INT(ZE/256):POKE 49166,ZE
      -INT(ZE/256)*256                            <211>
63117 PRINT"{HOME}"ZE:PRINT"{DOWN}GOTO6311
      9:"                                         <127>
63118 POKE 631,19:POKE 632,13:POKE 633,17:
      POKE 634,13:POKE 198,4:STOP                 <014>
63119 ZE=PEEK(49166)+256*PEEK(49165):ZE=ZE
      +1:IF ZE<63117 THEN 63116                   <173>
63120 PRINT"{CLR,2DOWN}"ZE:PRINT ZE+1:PRIN
      T ZE+2:PRINT ZE+3:PRINT ZE+4:PRINT"{
      HOME}";                                     <185>
63121 POKE 631,13:POKE 632,13:POKE 633,13:
      POKE 634,13:POKE 635,13:POKE 198,5          <094>
```

Listing 1. »Invisible«, der (fast) perfekte Listschutz (Schluß)

```
NAME : !GOSUB               C000 C0CA
-------------------------------------
C000 : A9 C0 8D 09 03 A9 0F 8D   63
C008 : 08 03 4C AE A7 EA EA EA   CE
C010 : 20 73 00 C9 21 F0 07 20   19
C018 : 79 00 4C E7 A7 EA E6 7A   04
C020 : D0 02 E6 7B EA A5 7A 85   EB
C028 : 22 A5 7B 85 23 A5 2B 85   C3
C030 : 7A A5 2C 85 7B 20 73 00   BF
C038 : C9 8F F0 0F A5 2E C5 7B   C1
C040 : D0 F3 A5 2D C5 7A D0 ED   68
C048 : 4C A2 C0 A0 00 20 73 00   F8
C050 : D1 22 D0 E1 C8 C0 28 F0   B8
C058 : EF A9 3A D1 22 D0 EE 20   89
C060 : 09 A9 18 98 65 7A 85 7A   BC
C068 : 90 02 E6 7B A0 04 B1 7A   08
C070 : 85 15 88 B1 7A 85 14 A5   48
C078 : 22 85 7A A5 23 85 7B A9   50
C080 : 03 20 FB A3 A9 A7 48 A9   53
C088 : AE 48 A5 7B 48 A5 7A 48   5F
C090 : A5 3A 48 A5 39 48 A9 8D   B1
C098 : 48 20 79 00 20 A3 A8 4C   A9
C0A0 : AE A7 A2 1A BD AF C0 20   AA
C0A8 : 16 E7 CA D0 F7 4C AE A7   6A
C0B0 : 0D 45 4C 42 41 4C 49 41   D9
C0B8 : 56 41 20 54 4F 4E 20 45   B4
C0C0 : 4E 49 54 55 4F 52 42 55   AE
C0C8 : 53 0D CA 10 E5 48 A2 05   8C
```

Listing 2. Programmieren Sie strukturiert mit »!GOSUB«

```
10 REM BEISPIELPROGRAMM                          <113>
20 :                                             <252>
30 :                                             <006>
40 !EINGABE:                                     <088>
50 :                                             <026>
60 !BERECHNUNG:                                  <200>
70 :                                             <046>
80 !AUSGABE:                                     <194>
90 :                                             <066>
95 :                                             <071>
100 END                                          <102>
110 :                                            <086>
120 :                                            <096>
130 REM EINGABE                                  <132>
140 PRINT"{CLR,3DOWN,3RIGHT}";                   <185>
150 INPUT "DURCHMESSER";D                        <182>
160 RETURN                                       <218>
170 :                                            <146>
180 REM BERECHNUNG                               <179>
190 A=(π*D↑2)/4                                  <140>
200 RETURN                                       <002>
210 :                                            <186>
220 REM AUSGABE                                  <032>
225 REM BASIC-BEFEHL MIT ANFUEHRUNGSZEICHE
    N !!!                                        <087>
230 !"PRINTA":                                   <114>
240 RETURN                                       <042>
250 :                                            <226>
260 REM "PRINTA"                                 <123>
270 PRINT"{CLR,3DOWN,3RIGHT}KREISFLAECHE A
    =";A;"MM↑2"                                  <135>
280 RETURN                                       <084>
```

Listing 3. Anwendungsbeispiel für Listing 2

Windows leichtgemacht

Da in den meisten Programmen Menüs benötigt werden, und ich das entsprechende Unterprogramm bis jetzt jedesmal neu und in Basic verfaßte (was schlechte Benutzerfreundlichkeit und langsame Ausführungsgeschwindigkeit nach sich zog), entschloß ich mich, ein universelles Menü-Programm als Basic-Erweiterung in Assembler zu schreiben. Listing 4 zeigt das Ergebnis:

Merkmale:

- beliebig auf dem Bildschirm plazierbares Fenster mit Überschrift.
- frei definierbare Abbruchtaste (zum Beispiel für Undo-Funktion).
- Auswahl des Menüpunktes mit den Cursortasten und »RETURN« oder (wahlweise) wird jedem Menüpunkt (soweit möglich) ein Buchstabe zugeordnet, durch den er direkt aufgerufen werden kann.
- wahlweise wird der Bildschirm nach Verlassen des Menüs wieder hergestellt (oder später mit dem Basic-Befehl »KILLM«). Listing 5 zeigt ein Anwendungsbeispiel in Form einer simulierten Textverarbeitung.

Handhabung:

Laden Sie das Programm »XMENU« absolut von Diskette (anschließend »NEW« eingeben) und starten es mit »SYS 50175«. Ab jetzt stehen Ihnen zusätzlich folgende zwei Befehle zur Verfügung.

- MENU <x>, <y>, <l>, <p>, <t>, <m>, <s>, <ar>, <rv>
- KILLM

Mit »MENU« wird das Menü dargestellt und ausgeführt. Hier die Bedeutung der Parameter:

<x> X-Koordinate. Bereich von 0 bis 35
<y> Y-Koordinate. Bereich von 1 bis 20
<l> Breite der Menüpunkte. Bereich von 1 bis (36-x)
<p> Anzahl der Menüpunkte. Bereich von 1 bis 11-int(y/2+.5)
<t> ASCII-Code der Abbruchtaste (möglichst nicht 65

bis 90, da sonst die Tastaturwahl beeinträchtigt wird).

<m> Auswahlmodus. Bereich von 0 bis 255.
0: keine Tastaturauswahl;
ungleich 0: Tastaturauswahl erlaubt.

<s> Temporärmodus. Bereich von 0 bis 255
0: Hintergrund wird nach Verlassen des Menüs nicht (sofort wieder hergestellt);
ungleich 0: Hintergrund wird sofort wieder hergestellt.

<ar> Eindimensionales Stringarray. Der Index ist der der Menü-Überschrift und zugleich die Basis für die Menüpunkte. Sollte das Array weniger als <p>+Index Elemente haben, so gibt es ein »Gewurschtel« auf dem Bildschirm.

<rv> Numerische Variable. Enthält nach dem Aufruf die Nummer des Menüpunktes oder Null bei einem Abbruch. Achtung! Sollte die Variable vor dem Aufruf »MENU« nicht definiert sein, so gibt es ein Wirrwarr auf dem Bildschirm, da die Arraytabellen verschoben wurden. Der Befehl »KILLM« (ohne Parameter) holt den vor dem Aufruf von »MENU« gespeicherten Bildschirm in den aktuellen Bildschirm (das gleiche passiert, wenn der Temporärmodus eingeschaltet ist).

Technische Details

Das Programm befindet sich nach dem (absoluten) Laden im Speicher von $C000 bis $C4E4. Nach dem Aufruf mit SYS 50175 wird der Basic-Befehls-Vektor ($0308/$0309) auf $C3FF verbogen. In der Zeropage wird, während das Menü dargestellt wird, der Bereich von $A3 bis $AF benützt. Außerdem werden der Bildschirm und zwei Arrays im Bereich von $D000 bis $D423 zwischengespeichert.

Der Assembler-Programmierer kann den Programmteil von $C3FF bis $C4E5 ignorieren. Die Parameter müssen nur in die Tabelle von $C3F6 bis $C3FE eingetragen und das Programm ab $C000 angesprungen werden. Der Befehl »KILLM« kann durch SEC, JSR $C39D ersetzt werden.

Um XMENU in einen anderen Bereich zu verschieben, sind folgende Schritte nötig:

1. Der Bereich von $C000 bis $C4E4 muß nach der neuen Anfangsadresse verschoben werden (im folgenden Basis genannt).
2. Alle 3-Byte-Befehle, die sich auf den Bereich von $C000 bis $C4E4 beziehen, müssen an den Bereich von der Basis bis Basis +$4E4 angeglichen werden. (V-Befehl bei SMON).
3. Disassemblieren Sie den Bereich <Basis+$3FF> bis <Basis+$402>. Ersetzen Sie nun den Parameter des LDA-Befehls durch das Low-Byte und den Parameter des LDY-Befehls durch das High-Byte von <Basis+$40A>.
4. Disassemblieren Sie den Bereich <Basis+$40D> bis <Basis+$410>. Ersetzen Sie (wie in 3.) die entsprechenden Parameter der LDA-/LDY-Befehle durch das Low- beziehungsweise High-Byte von <Basis+$4DA>.
5. Disassemblieren Sie den Bereich <Basis+$41D> bis <Basis+$420>. Ersetzen Sie nun (wie in 3. und 4.) die Parameter der LDA-/LDY-Befehle durch das Low-/High-Byte von <Basis+$4DF>.

Die Initialisierungsadresse entspricht der Basis+$3FF.

Die einzelnen Routinen

Bezeichnung	Abstand zur Basis	Beschreibung
MENU	$0000=0	Maschinenroutine zum Darstellen des Menüs
KILLM	$039D=925	Danach: Angewählter Menüpunkt im Akku. Carry gelöscht: Hintergrund wird gespeichert. Carry gesetzt: Hintergrund wird wieder hervorgeholt.
X	$03F6=1015	X-Koordinate
Y	$03F7=1016	Y-Koordinate
L	$03F8=1017	Breite
P	$03F9=1018	Anzahl M.-Punkte
T	$03FA=1019	Abbruchtaste
M	$03FB=1020	Auswahlmodus
S	$03FC=1021	Temporärmodus
AR/L	$03FD=1022	Low-Byte des Zeigers auf Array-Tabelle (T.Format: Länge, Low-/High-Byte Adr. String).
AR/H	$03FE=1023	High-Byte des Zeigers auf Array-Tabelle

Um sich mit den Fähigkeiten des Programms vertraut zu machen, sollten Sie unbedingt das Demo-Programm durchstudieren.

(Daniel Buchheit/tr)

```
NAME : XMENU              C000 C4E5
-----------------------------------
C000 : 18 20 9D C3 20 1A C1 A9   35
C008 : 01 85 A5 A9 00 85 AB 20   85
C010 : D9 C0 A5 C6 D0 0E 20 FE   87
C018 : C0 A5 C6 D0 07 20 CA C0   95
C020 : A5 C6 F0 03 20 37 C0 A5   CF
C028 : AB F0 E7 AD FC C3 F0 04   B5
C030 : 38 20 9D C3 A5 A5 60 A9   B4
C038 : 00 85 A7 AD 77 02 C9 C0   CB
C040 : 90 02 29 7F CD FA C3 D0   71
C048 : 08 A2 00 86 A5 A2 01 86   F3
C050 : AB C9 41 90 2E C9 80 B0   D7
C058 : 2A C9 5B 90 04 C9 61 90   85
C060 : 22 AD FB C3 F0 1D AD 77   6E
C068 : 02 20 51 C3 AA 18 20 9D   6E
C070 : C2 F0 10 85 A5 A5 A6 10   A1
C078 : 03 20 FE C0 A9 01 85 AB   73
C080 : 20 D9 C0 AD 77 02 C9 0D   3B
C088 : D0 04 A2 01 86 AB C9 91   33
C090 : D0 04 A2 FF 86 A7 C9 11   FA
C098 : D0 04 A2 01 86 A7 A5 A7   BF
C0A0 : F0 11 18 A5 A5 65 A7 85   03
C0A8 : A5 A5 A6 10 03 20 FE C0   7A
C0B0 : 20 D9 C0 A5 AB F0 0E A5   67
C0B8 : A6 30 03 20 FE C0 A5 A5   13
C0C0 : D0 03 20 FE C0 A9 00 85   5E
C0C8 : C6 60 A0 73 A2 00 A5 C6   A3
C0D0 : D0 06 CA D0 F9 88 D0 F4   81
C0D8 : 60 A9 00 85 A6 A5 A5 D0   8D
C0E0 : 05 AD F9 C3 85 A5 CD F9   63
C0E8 : C3 90 06 F0 04 A9 01 85   30
C0F0 : A5 18 0A 6D F7 C3 AA AC   73
C0F8 : F6 C3 18 4C 0A E5 A4 D3   69
C100 : C8 C8 A2 00 A5 A6 49 80   8B
C108 : 85 A6 B1 D1 29 7F 05 A6   77
C110 : 91 D1 C8 E8 EC F8 C3 D0   20
C118 : F1 60 20 28 C1 AD FB C3   47
C120 : F0 03 4C B1 C1 4C 4D C2   14
C128 : AE F7 C3 CA AC F6 C3 C8   3F
C130 : 18 20 0A E5 A9 00 20 72   98
C138 : C2 A5 A7 AA E8 E8 A9 A4   D2
C140 : 20 5A C2 20 80 C3 A9 B0   70
C148 : 20 D2 FF A9 A5 20 D2 FF   AD
C150 : 20 65 C2 A9 A7 20 D2 FF   CF
C158 : 38 AD F8 C3 E5 A7 AA A9   B7
C160 : 60 20 5A C2 A9 6E 20 6A   23
C168 : C3 20 80 C3 A9 7D 20 D2   80
C170 : FF A9 A3 A6 A7 E8 E8 20   A7
C178 : 5A C2 38 AD F8 C3 E5 A7   8C
C180 : AA A9 20 20 5A C2 A9 5D   28
C188 : 20 6A C3 18 AD F9 C3 0A   9F
C190 : 6D F7 C3 69 02 AA AC F6   2D
C198 : C3 18 20 0A E5 A9 AD 20   53
C1A0 : D2 FF A9 60 AE F8 C3 E8   7C
C1A8 : E8 20 5A C2 A9 7D 4C 6A   1C
C1B0 : C3 20 3F C3 A9 00 85 AD   D8
C1B8 : A9 01 85 AB 85 AC A5 AC   66
C1C0 : 20 72 C2 A5 A7 D0 03 4C   24
C1C8 : FD C1 A4 AB 20 66 C3 20   C9
C1D0 : 51 C3 B0 15 18 AA 20 9D   64
C1D8 : C2 D0 0E 38 A5 AC 20 9D   08
C1E0 : C2 A6 AB 20 D5 C2 4C F3   71
C1E8 : C1 38 A6 AD E6 AD A5 AC   F0
C1F0 : 20 A5 C2 A5 AC E6 AC CD   98
C1F8 : F9 C3 90 C2 2C 38 60 A5   A1
C200 : AD F0 FB A9 01 85 AC A9   9C
C208 : 01 85 AB 18 A6 AC CA 20   F5
C210 : A5 C2 85 AE 20 72 C2 A5   39
C218 : AB C5 A7 F0 20 E6 AB A4   DF
C220 : AB 20 66 C3 20 51 C3 B0   EA
C228 : EE 85 AF 18 AA 20 9D C2   6F
C230 : D0 E5 38 A5 AE A6 AF 20   D5
C238 : 9D C2 A6 AB 2C A2 00 A5   78
C240 : AE 20 D5 C2 A5 AC E6 AC   81
C248 : C5 AD 90 BB 60 AC F9 C3   5A
C250 : A2 00 98 20 D5 C2 88 D0   54
C258 : F9 60 E0 00 F0 06 20 D2   1F
C260 : FF CA D0 FA 60 A0 00 B1   C6
C268 : A5 20 D2 FF C8 C4 A7 D0   C5
C270 : F6 60 85 A3 18 0A 65 A3   1B
C278 : 6D FD C3 85 A3 A9 00 6D   E8
C280 : FE C3 85 A4 A0 00 B1 A3   6E
C288 : CD F8 C3 90 03 AD F8 C3   DD
C290 : 85 A7 C8 B1 A3 85 A5 C8   E0
C298 : B1 A3 85 A6 60 08 20 C7   A7
C2A0 : C2 A9 00 F0 06 08 20 C7   06
C2A8 : C2 A9 1A 85 A3 A9 D0 85   4C
C2B0 : A4 28 20 74 C3 A5 A8 A4   54
C2B8 : A9 B0 02 B1 A3 91 A3 85   D0
C2C0 : A8 20 7A C3 4C CE C2 85   E1
C2C8 : A8 86 A9 84 AA 60 A4 AA   44
C2D0 : A6 A9 A5 A8 60 20 C7 C2   75
C2D8 : 18 A5 A8 0A 6D F7 C3 AA   29
C2E0 : AC F6 C3 18 20 0A E5 20   25
C2E8 : 2B C3 A5 A8 20 72 C2 38   84
C2F0 : AD F8 C3 E5 A7 85 A3 20   3C
C2F8 : 65 C2 A9 2E A6 A3 20 5A   AB
C300 : C2 20 35 C3 20 80 C3 20   ED
C308 : 2B C3 A6 A9 F0 0B CA A9   D9
C310 : 20 20 5A C2 A9 BB 20 D2   B6
C318 : FF 38 AD F8 C3 E5 A9 AA   25
C320 : A9 20 20 5A C2 20 35 C3   B6
C328 : 4C CE C2 A9 7D 20 D2 FF   E5
C330 : A9 20 4C D2 FF A9 20 20   65
C338 : D2 FF A9 5D 4C 6A C3 20   87
C340 : 74 C3 A9 00 A2 1A CA 9D   62
C348 : 00 D0 E0 00 D0 F8 4C 7A   E3
C350 : C3 29 7F C9 7B B0 0D C9   C6
C358 : 41 90 09 29 1F E9 01 C9   22
```

Listing 4. Die Befehlserweiterung »XMENU«

```
C360 : 1A B0 01 24 38 60 88 B1  A3
C368 : A5 60 A4 D3 91 D1 AD 86  4C
C370 : 02 91 F3 60 78 A9 30 85  E4
C378 : 01 60 A9 37 85 01 58 60  7D
C380 : A9 28 AE F6 C3 A4 D3 86  85
C388 : D3 C0 28 90 02 A9 50 18  B6
C390 : 65 D1 85 D1 A5 D2 69 00  10
C398 : 85 D2 E6 D6 60 20 74 C3  7B
C3A0 : A2 00 B0 28 BD 00 04 9D  9B
C3A8 : 24 D0 BD 00 05 9D 24 D1  15
C3B0 : BD 00 06 9D 24 D2 BD 00  72
C3B8 : 07 9D 24 D3 E8 D0 E5 A5  09
C3C0 : D3 A4 D6 8D 0C D4 8C 0D  00
C3C8 : D4 4C 7A C3 BD 24 D0 9D  55
C3D0 : 00 04 BD 24 D1 9D 00 05  DA
C3D8 : BD 24 D2 9D 00 06 BD 24  7F
C3E0 : D3 9D 00 07 E8 D0 E5 AC  69
C3E8 : 0C D4 AE 0D D4 20 7A C3  6B
C3F0 : 18 4C 0A E5 4A 4A 00 01  67
C3F8 : 24 0A 85 01 00 66 9F A9  A8
C400 : 0A A0 C4 8D 08 03 8C 09  1A
C408 : 03 60 20 73 00 A9 DA A0  AC
C410 : C4 85 A3 84 A4 20 37 C4  C2
C418 : D0 03 4C 55 C4 A9 DF A0  82
C420 : C4 85 A3 84 A4 20 37 C4  D2
C428 : D0 07 38 20 9D C3 4C AE  14
C430 : A7 20 79 00 4C E7 A7 A0  2A
C438 : 00 B1 A3 F0 09 D1 7A D0  C2
C440 : 04 C8 4C 39 C4 60 18 98  C3
C448 : 65 7A 85 7A A9 00 65 7B  C2
C450 : 85 7B A9 00 60 A0 00 20  49
C458 : CF C4 8A 99 F6 C3 C8 C0  91
C460 : 07 D0 F4 20 8B B0 A0 04  D9
C468 : B1 5F C9 01 D0 59 A5 47  58
C470 : A4 48 8D FD C3 8C FE C3  7F
C478 : 20 FD AE AD F6 C3 C9 24  F5
C480 : B0 48 AD F7 C3 F0 43 C9  23
C488 : 15 B0 3F 38 A9 24 ED F8  32
C490 : C3 30 37 CD F6 C3 90 32  27
C498 : AD F8 C3 F0 2D 18 AD F9  0E
C4A0 : C3 F0 27 0A 8D AB C4 38  A0
C4A8 : A9 16 E9 00 CD F7 C3 90  A3
C4B0 : 19 20 8B B0 85 49 84 4A  1B
C4B8 : 20 00 C0 A8 20 A2 B3 A5  4F
C4C0 : 0E 20 C2 A9 4C AE A7 A2  E2
C4C8 : 0B 2C A2 0E 4C 37 A4 84  6E
C4D0 : A3 20 9E B7 20 FD AE A4  18
C4D8 : A3 60 4D 45 4E 55 00 4B  CD
C4E0 : 49 4C 4C 4D 00 CE 20 78  F4
```

Listing 4. (Schluß)

```
0 SYS 57812"XMENU",8,1:POKE 780,0:SYS 6549
  3:REM XMENU LADEN                          <112>
5 SYS 50175:AM=0:ME=0:PT=1                   <224>
10 GOSUB 1100:GOSUB 1000:REM INIT            <061>
11 :                                         <243>
15 MT$="MAIN":GOSUB 1200                     <018>
20 MENU 9,5,17,6,141,1,0,M$(0),AM            <156>
21 :                                         <253>
22 IF AM<6 THEN:KILLM                        <188>
23 IF AM=0 THEN GOSUB 1220:GOSUB 1022:GOTO
    15                                       <029>
24 IF AM=1 THEN MD$=M$(AM):GOSUB 1250:GOTO
    20                                       <220>
25 IF AM=6 THEN GOSUB 1220:POKE 214,23:SYS
    58732:END                                <182>
26 ON AM-1 GOTO 40,80,100,140                <164>
30 :                                         <006>
36 :                                         <012>
40 GOSUB 1280                                <029>
41 MT$="PRINT":GOSUB 1200                    <189>
43 MENU 9,5,17,5,141,1,1,P$(0),AM            <053>
44 :                                         <020>
48 IF AM=0 THEN GOSUB 1220:GOSUB 1265:GOTO
    20                                       <043>
51 IF AM=1 THEN 60                           <074>
54 MD$=P$(AM):GOSUB 1265:GOSUB 1250:KILLM:
   GOTO 43                                   <142>
55 :                                         <031>
60 MT$="PTYPE":GOSUB 1200                    <130>
65 MENU 9,5,17,5,141,1,1,PT$(0),AM           <091>
70 IF AM=0 THEN GOSUB 1220:GOTO 43           <200>
73 PT=AM:GOSUB 1265:GOSUB 1280:GOTO 65       <202>
74 :                                         <050>
80 MT$="EDIT":GOSUB 1200                     <106>
83 MENU 9,5,17,5,141,1,1,E$(0),AM            <215>
86 :                                         <062>
89 IF AM=0 THEN GOSUB 1220:GOTO 20           <215>
92 MD$=E$(AM):GOSUB 1250:GOTO 83             <065>
93 :                                         <069>
100 MT$="DISK":GOSUB 1200                    <243>
105 MENU 9,5,17,6,141,1,1,D$(0),AM           <141>
108 :                                        <084>
110 IF AM=0 THEN GOSUB 1220:GOTO 20          <236>
112 IF AM=2 THEN 120                         <134>
114 MD$=D$(AM):GOSUB 1250:GOTO 105           <148>
115 :                                        <091>
120 MT$="DCMDS":GOSUB 1200                   <207>
123 MENU 9,5,17,4,141,1,1,DC$(0),AM          <094>
126 :                                        <102>
130 IF AM=0 THEN GOSUB 1220:GOTO 105         <233>
135 MD$=DC$(AM):GOSUB 1250:GOTO 123          <239>
136 :                                        <112>
140 MT$="TEXT":GOSUB 1200                    <077>
145 MENU 9,5,17,5,141,1,1,T$(0),AM           <157>
146 :                                        <122>
150 IF AM=0 THEN GOSUB 1220:GOTO 20          <020>
155 MD$=T$(AM):GOSUB 1250:GOTO 145           <065>
999 :                                        <213>
1000 REM BILDSCHIRM                          <115>
1001 :                                       <215>
1002 PRINT CHR$(14)"{CLR,CYAN,RVSON}DEMOWR
     ITE NAME:DEMO.TEX{2SPACE}ST 1 ZL 16 S
     P 5                                     <179>
1003 PRINT"{RVSON}HIERARCHIE:{29SPACE,RVOF
     F}"                                     <190>
1004 PRINT"{3SPACE}DIES IST EINE TEXTVERAR
     BEITUNGS-AT-                            <254>
1005 PRINT" TRAPPE UM IHNEN DIE ANWENDUNG
     DES MENU-";                             <022>
1006 PRINT" UNTERPROGRAMMS IN DER PRAXIS Z
     U ZEIGEN.                               <107>
1007 PRINT"{3SPACE}IN DAS HAUPTMENU GELANG
     EN SIE MIT                              <237>
1008 PRINT" CTRL-M. IN DAS NACHSTE MENU KO
     MMEN SIE                                <011>
1009 PRINT" MIT RETURN ODER DURCH DRUCKEN
     EINES UN-";                             <098>
1010 PRINT" TERLEGTEN BUCHSTABENS. UM IN D
     AS VOR-                                 <074>
1011 PRINT" HERGEHENDE MENU ZU KOMMEN MUSS
     EN SIE                                  <034>
1012 PRINT" SHIFT-RETURN DRUCKEN. DIE MENU
     -HIERAR-                                <225>
1013 PRINT" CHIE, DIE ANZEIGT IN WELCHER M
     ENU-EBENE";                             <073>
1014 PRINT" SIE SICH BEFINDEN, SEHEN SIE I
     N DER                                   <009>
1015 PRINT" DRITTEN BILDSCHIRMZEILE.{DOWN} <054>
1016 PRINT"{3SPACE}NACHDEM SIE EINE FUNKTI
     ON ANGEWAHLT                            <022>
1017 PRINT" HABEN, ERSCHEINT DIE MELDUNG "
     CHR$(34)"FUNKTION                       <038>
1018 PRINT" NICHT IMPLEMENTIERT"CHR$(34)"
     UND DIE AUFFORDER-";                    <176>
1019 PRINT" UNG EINE TASTE ZU DRUCKEN. WEN
     N SIE                                   <115>
1020 PRINT" JETZT DIREKT IN DEN TEXTMODUS
     WOLLEN,                                 <208>
1021 PRINT" MUSSEN SIE '←' DRUCKEN."         <016>
1022 :                                       <236>
1023 REM AUF CTRL-M WARTEN                   <168>
1024 :                                       <240>
1025 POKE 211,24:POKE 214,23:SYS 58732       <193>
1027 POKE 204,0                              <166>
1030 WAIT 198,1:POKE 198,0:IF PEEK(631)<>1
     3 THEN 1030                             <059>
1035 WAIT 207,1,1:POKE 204,1                 <041>
1040 REM FERTIG                              <038>
1045 RETURN                                  <087>
1046 :                                       <006>
1100 REM TEXTE                               <074>
1101 :                                       <061>
1105 DIM M$(6),P$(5),E$(7),D$(6),T$(5),DC$
     (4),PT$(5)                              <148>
1110 M$(0)="MAIN MENU"                       <153>
1111 M$(1)="NEW DOCUMENT"                    <017>
1112 M$(2)="PRINT"                           <109>
1113 M$(3)="EDIT FUNCTIONS"                  <067>
1114 M$(4)="DISK HANDLING"                   <060>
1115 M$(5)="TEXT PARAMETERS"                 <019>
1116 M$(6)="QUIT DEMOWRITE"                  <227>
1117 :                                       <077>
1120 P$(0)="PRINT MENU"                      <078>
1121 P$(1)="TYPE OF PRINTER"                 <227>
1122 P$(2)="SET JUSTIFICATION"               <072>
1123 P$(3)="FROM/TO PAGE"                    <026>
1124 P$(4)="DEFINE PAPER"                    <043>
1125 P$(5)="PRINT TEXT"                      <189>
1126 :                                       <086>
1130 E$(0)="EDIT MENU"                       <234>
1131 E$(1)="FIND"                            <194>
1132 E$(2)="DELETE"                          <119>
1133 E$(3)="REPLACE"                         <020>
```

Listing 5. Eine simulierte Textverarbeitung als Anwendungsbeispiel für »XMENU«

```
1134 E$(4)="COPY"                          <152>
1135 E$(5)="MOVE"                          <188>
1136 :                                     <096>
1140 D$(0)="DISK MENU"                     <193>
1141 D$(1)="DIRECTORY"                     <096>
1142 D$(2)="COMMANDS OF DISK"              <045>
1143 D$(3)="IMPORT VIZA-TEXT"              <040>
1144 D$(4)="IMPORT SEQ-TEXT"               <076>
1145 D$(5)="EXPORT SEQ-TEXT"               <145>
1146 D$(6)="SAVE TEXT"                     <034>
1147 :                                     <107>
1150 T$(0)="TEXT MENU"                     <212>
1151 T$(1)="TEXT WIDTH"                    <005>
1152 T$(2)="RENAME TEXT"                   <130>
1153 T$(3)="CHARACTER SET"                 <140>
1154 T$(4)="SCREEN COLOR"                  <087>
1155 T$(5)="STORAGE DEVICE"                <113>
1156 :                                     <116>
1160 DC$(0)="DCOMMAND MENU"                <162>
1161 DC$(1)="RENAME FILE"                  <036>
1162 DC$(2)="DELETE FILE"                  <079>
1163 DC$(3)="FORMAT DISK"                  <087>
1164 DC$(4)="VALIDATE DISK"                <001>
1165 :                                     <125>
1170 PT$(0)="PRINTER MENU"                 <080>
1171 PT$(1)="ITOH 8510"                    <034>
1172 PT$(2)="EPSON RX/FX"                  <200>
1173 PT$(3)="VC 1526"                      <024>
1174 PT$(4)="BROTHER HR-5C"                <024>
1175 PT$(5)="STAR DELTA 10"                <139>
1176 :                                     <136>
1177 RETURN                                <219>
1178 :                                     <138>
1200 REM HIERARCHIE ERWEITERN              <102>
1201 :                                     <161>
1205 PRINT"{HOME,2DOWN,RVSON}"TAB(12+7*ME)
     "->";MT$                              <178>
1210 ME=ME+1:RETURN                        <122>
1211 :                                     <171>
1220 REM HIERARCHIE ZURUCKSETZEN           <034>
1221 :                                     <181>
1225 PRINT"{HOME,2DOWN,RVSON}"TAB(5+7*ME)"
     {7SPACE}"                             <108>
1230 ME=ME-1:RETURN                        <014>
1231 :                                     <191>
1250 REM MELDUNG AUSGEBEN                  <037>
1251 :                                     <211>
1255 PRINT"{HOME,DOWN}"CHR$(34);MD$;CHR$(3
     4)" NICHT IMPL. - TASTE!";            <138>
1260 POKE 198,0:WAIT 198,1:POKE 198,0:IF P
     EEK(631)=95 THEN RUN                  <095>
1265 PRINT"{HOME,DOWN,40SPACE}";           <232>
1270 RETURN                                <056>
1271 :                                     <231>
1275 REM AKTUELLEN DRUCKER AUSGEBEN        <206>
1276 :                                     <236>
1280 PRINT"{HOME,DOWN}CURRENT SELECTED PRI
     NTER:";PT$(PT)                        <127>
1285 RETURN                                <073>
```

Listing 5. Beispiel zu »XMENU« (Schluß)

Das besondere Turbo Tape

»Extratape« ist ein Schnellladeprogramm für den C64 und Datasette. Es erlaubt jedem, ohne aufwendiges Herumspulen, seine Programme jederzeit mit 8- bis 9facher Geschwindigkeit zu laden.

Das Besondere an »Extratape« ist, daß man zum Laden des Programms das Extratape-Programm nicht mehr benötigt! Die schnelle Laderoutine ist nämlich als kurzer Vorspann direkt vor dem gespeicherten Programm vorhanden. Man kann ein so behandeltes Programm also ganz normal mit »LOAD« laden.

Nachdem man das Maschinenprogramm (Listing 6) mit dem MSE eingegeben und auf Kassette oder Diskette gespeichert hat, muß man es wie ein Basic-Programm laden und mit »RUN« starten. Nach dem Start wird vom Programm der Save-Vektor ($0332/0333) auf eine eigene Save-Routine bei $CDE0 verbogen. Diese Routine simuliert ein neues Gerät mit der Gräteadresse 7, nämlich die Fasttape-Datasette. Speichert man nun ein Programm mit der Geräteadresse 7, zum Beispiel »SAVE "TEST",7«, so erscheint die Aufforderung, <Record> und <Play> zu drücken, und das Programm wird mit ungefähr 9facher Normalgeschwindigkeit auf Band geschrieben.

Man sollte übrigens bei Benutzung von »Extratape« keine Superbillig-Kassetten (Kaufhaus-Sonderangebot Wühltisch für 98 Pfennige) verwenden, da sonst häufig Ladefehler auftreten können. Diese werden durch einen »I/O ERROR« am Ende des Ladevorgangs angezeigt. Ladefehler können auch durch das Ein-/Ausschalten großer Elektrogeräte in der Nähe des Computers (zum Beispiel Staubsauger, Elektroheizung im gleichen Raum und so weiter) auf das Band gebracht werden. Deshalb empfiehlt es sich, bei wichtigen Programmen nach dem Speichern eine Kontrolle (durch Laden in den Computer) durchzuführen.

Die gespeicherten Programme laufen zwar trotz angezeigtem Fehler meist einwandfrei, aber es ist dennoch unschön, einen Fehler auf der Kassette zu haben. Wem diese Fehler zu häufig auftreten, obwohl die Programme immer laufen, der kann die Fehlerabfrage mit »POKE 52951,208« und »POKE 52952,16« (vor dem Speichern eines Programms) abschalten. Eventuell auftretende Fehler werden nun nicht mehr angezeigt.

Der Grund für diese relativ hohe Fehleranfälligkeit ist, daß in einer Sekunde ungefähr 3830 Impulse (Bit) auf das Band geschrieben werden. Und das wären dann 425 Byte in einer Sekunde! (Es werden neun Bit für ein ganzes Byte benötigt, da immer ein Bit zur Synchronisation dient.) Mit dieser Geschwindigkeit übertrifft man sogar die Floppy (zirka 300 Byte in der Sekunde).

Es fällt auch auf, daß nach dem SAVEn eines Programmes die Schriftfarbe immer auf Schwarz steht. Dieser Effekt kommt dadurch zustande, daß die spätere Laderoutine beim Speichern an den Namen des Programmes angehängt wird, und so das Ladeprogramm als wirrer Zeichensalat hinter dem »SAVING« kurz auf dem Bildschirm erscheint.

Mit einigen POKEs ist es auch möglich, ein Maschinenprogramm absolut (also an seiner Originaladresse) zu speichern: Man muß nur die Zeiger auf Programmstart und -ende in den Speicherstellen 43/44 (Start) und 45/46 (Ende) auf Start und Ende des Maschinenprogramms einstellen. Man muß allerdings aufpassen, daß »Extratape« ($CDE0 bis $CFE0) nicht überschrieben wird (Das gäbe beim nächsten SAVE-Befehl einen richtig schönen Systemabsturz...). Auch bei normalen Programmen sollte man darauf achten, daß dieser Bereich nicht gelöscht wird. Extratape läuft übrigens einwandfrei mit vielen Floppy-Speedern zusammen.

(P. Plamper/tr)

```
NAME : EXTRATAPE          0801 0A41
-----------------------------------
0801 : 15 08 0A 00 9E 32 30 38  4A
0809 : 30 20 45 58 54 52 41 54  2B
0811 : 41 50 45 00 00 00 FF 50  6C
0819 : 50 20 31 39 38 35 20 A2  E0
0821 : 00 BD 40 08 9D E0 CD BD  A4
0829 : 40 09 9D E0 CE E8 D0 F1  CD
0831 : A9 E0 A2 CD 8D 32 03 8E  40
0839 : 33 03 4C 74 A4 EA EA A5  28
0841 : BA C9 07 F0 03 4C ED F5  F6
0849 : A2 10 A9 20 9D 41 CE CA  16
0851 : 10 F8 A5 B7 F0 0C 29 0F  70
0859 : A8 88 B1 BB 99 41 CE 88  19
0861 : 10 F8 A9 70 A2 03 8D 02  E2
0869 : 03 8E 03 03 A9 00 A2 01  FC
0871 : A0 01 20 BA FF A9 BF A2  83
0879 : 41 A0 CE 20 BD FF A9 00  44
0881 : A2 03 85 FB 86 FC A9 FB  74
0889 : A2 05 A0 03 20 D8 FF 20  3F
0891 : 44 E5 20 00 CF A9 B3 A2  6E
0899 : A4 8D 02 03 8E 03 03 60  B3
```

Listing 6. »Extratape«, ein Turbo-Tape-Programm mit Raffinessen.

```
08A1 : 00 00 00 00 00 00 00 00  A2
08A9 : 00 00 00 00 00 00 00 00  AA
08B1 : 00 A9 00 85 AB A9 10 2C  D7
08B9 : 0D DC F0 FB AD 0D DD A2  F0
08C1 : 11 8E 0E DD EE 20 D0 4A  20
08C9 : 26 AB 90 E9 A5 AB 60 78  50
08D1 : A9 7F 8D 0D DD A9 00 8D  85
08D9 : 11 D0 85 AA 8D 04 DD A2  BF
08E1 : 01 8E 05 DD A2 06 86 01  9D
08E9 : A9 83 A2 A4 8D 02 03 8E  A3
08F1 : 03 03 EA 20 56 03 D0 FB  ED
08F9 : 20 52 03 F0 FB C9 2C D0  82
0901 : F2 A0 03 20 52 03 99 AC  05
0909 : 00 88 10 F7 A0 00 20 52  80
0911 : 03 91 AC 45 AA 85 AA 20  72
0919 : DB FC 20 D1 FC 90 ED 20  01
0921 : 52 03 48 A9 37 85 01 A9  33
0929 : 1B 8D 11 D0 A9 01 85 C0  A3
0931 : 58 68 C5 AA F0 12 A6 D6  6C
0939 : CA 86 D6 20 6C E5 A0 00  78
0941 : 20 2B F1 A9 0D 20 D2 FF  C5
0949 : 20 AA F5 86 2D 84 2E 4C  55
0951 : 83 A4 00 44 41 54 45 20  BB
0959 : 31 33 2C 34 2C 38 36 20  53
0961 : 38 F8 78 A9 7F 8D 0D DD  BD
0969 : A9 80 A2 02 8D 04 DD 8E  C9
0971 : 05 DD A9 00 8D 11 D0 85  7F
0979 : AA AA A8 A9 06 85 01 CA  FE
0981 : D0 FD 88 D0 FA A9 19 8D  08
0989 : 0E DD A2 03 B5 2B 95 AC  F3
0991 : CA 10 F9 A0 00 A9 00 20  83
0999 : C4 CF 88 D0 F8 A9 00 20  9E
09A1 : C4 CF 88 D0 F8 A9 00 20  A6
09A9 : C4 CF 88 D0 F8 A9 2C 20  5F
09B1 : C4 CF EA A0 03 B9 AC 00  DC
09B9 : 20 C4 CF 88 10 F7 EA EA  82
09C1 : EA A0 00 B1 AC 45 AA 85  DC
09C9 : AA B1 AC 20 C4 CF 20 DB  7E
09D1 : FC 20 D1 FC 90 EB EA EA  DB
09D9 : EA A5 AA 20 C4 CF EA EA  91
09E1 : EA A9 00 8D 0E DD A9 1B  FE
09E9 : 8D 11 D0 A9 37 85 01 A9  5F
09F1 : 01 85 C0 58 60 EA EA A5  44
09F9 : 01 09 08 85 01 A2 05 CA  00
0A01 : D0 FD 29 F7 85 01 60 AD  56
0A09 : 0D DD 29 01 F0 F9 A9 00  F5
0A11 : 26 AB 90 02 A9 01 8D 05  54
0A19 : DD A9 19 8D 0E DD 4C 98  F5
0A21 : CF EA EA 48 A9 80 85 AB  35
0A29 : 20 AB CF 68 85 AB A9 08  0B
0A31 : 8D FF CF 20 AB CF CE 20  3A
0A39 : D0 CE FF CF D0 F5 60 00  AB
```

Listing 6. (Schluß)

Spitzengrafiken auf MPS-802

Mit dem Programm »Support-802« wird es auch den MPS-802-Benutzern ermöglicht, die wirklich erstklassige Basic-Erweiterung »MPS-Support« zu benutzen (Auflösung: 640 x 400 Punkte!). Die Druckroutine hat dieselben Funktionen wie die in der 64'er, Ausgabe 2/86, veröffentlichte. Die einzelnen Befehle und ihre Bedeutung entnehmen Sie deshalb bitte dem 64'er-Heft.

Beim Abtippen gehen Sie wie folgt vor:

1. Falls Sie es noch nicht getan haben, tippen Sie das Programm »MPS-Support« aus der 64'er, Ausgabe 2/86, mit Hilfe des MSE ab und speichern es auf Diskette.
2. Jetzt tippen Sie Listing 7 mit dem MSE ab und speichern es unter dem Namen »dump 802« auf Diskette.
3. Laden Sie jetzt bitte »dump 802« absolut (also ,8,1) und geben Sie »NEW« ein.
4. Nun laden Sie MPS-Support aus Ausgabe 2/86.
5. Wenn Sie »SYS 49152« eingeben, wird die neue DUMP-Routine ins MPS-Support integriert und das ganze Programm unter dem Namen »support 802« auf Diskette gespeichert. Es ist jetzt ein Block länger geworden, aber Sie können es wie gewohnt laden und starten.

In Bild 1 sehen Sie eine Netzgrafik, die mit »support 802« ausgedruckt wurde.

Die Programmservice-Diskette enthält natürlich das fertige Programm »support 802«. (Oliver Koch/tr)

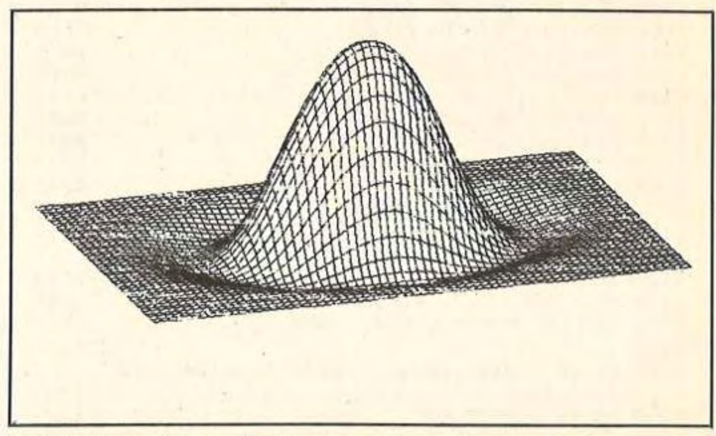


Bild 1. Diese Netzgrafik (verkleinert) wurde auf einem MPS-802 ausgegeben.

Nachfolgend eine kurze Befehlsübersicht. Die genaue Bedeutung finden Sie in der Ausgabe 2/86 ab Seite 59.

GRAPHIC / TEXT / WINDOW A,B / UPDATE / ZOOM
INCREASE A,B / RESTART / CLEAR / COLOUR A,B,C /
DOT A,B / CDOT A,B / LINE A,B,C,D / CLINE A,B,C,D /
TEST A,B / PATTERN A / CHAR A,B,C / DUMP
POLY A,B,C,D,E,F,G,H
CPOLY A,B,C,D,E,F,G,H
FPOLY A,B,C,D,E,F,G,H
EPOLY A,B,C,D,E,F,G,H

```
NAME : DUMP 802          C000 C2AC
-----------------------------------
C000 : A9 17 85 5F A9 C1 85 60  02
C008 : A9 1F 85 5A A9 C2 85 5B  6B
C010 : A9 D8 85 58 A9 12 85 59  86
C018 : 20 BF A3 A9 1F 85 5F A9  25
C020 : C2 85 60 A9 AC 85 5A A9  A6
C028 : C2 85 5B A9 00 85 58 A9  9A
C030 : 18 85 59 20 BF A3 A9 97  54
C038 : 85 5F A9 C0 85 60 A9 16  1D
C040 : 85 5A A9 C1 85 5B A9 73  56
C048 : 85 58 A9 17 85 59 20 BF  6A
C050 : A3 A9 4C 8D 10 08 A9 F4  5E
C058 : 8D 11 08 A9 16 8D 12 08  CB
C060 : A9 01 85 FB A9 08 85 FC  55
C068 : A9 01 A2 08 A0 01 20 BA  43
C070 : FF A9 0B A2 8C A0 C0 20  6C
C078 : BD FF A9 FB A2 00 A0 18  FB
C080 : 20 D8 FF A9 01 20 C3 FF  61
C088 : 20 CC FF 60 53 55 50 50  DC
C090 : 4F 52 54 2D 38 30 32 A9  E4
C098 : 8E 20 D2 FF A9 20 8D E8  8E
C0A0 : CF A9 CC 8D E9 CF A9 FF  EC
C0A8 : 8D EA CF 8D ED CF EA 8D  74
C0B0 : EB CF A9 DD 8D EC CF A9  7C
C0B8 : 20 8D EE CF A9 D2 8D EF  9C
C0C0 : CF A9 FF 8D F0 CF A9 18  7A
C0C8 : 8D F1 CF A9 A5 8D F2 CF  A9
C0D0 : A9 FB 8D F3 CF A9 69 8D  64
C0D8 : F4 CF A9 80 8D F5 CF A9  49
C0E0 : 85 8D F6 CF A9 FB 8D F7  84
C0E8 : CF A9 A5 8D F8 CF A9 FC  55
C0F0 : 8D F9 CF A9 69 8D FA CF  31
C0F8 : A9 02 8D FB CF A9 85 8D  01
C100 : FC CF A9 FC 8D FD CF A9  49
C108 : 60 8D FE CF A9 15 8D 8C  7B
C110 : 0A 4C 15 08 4C 15 08 A9  68
C118 : 32 A2 04 A0 00 20 A4 17  72
C120 : A9 33 A2 04 A0 05 20 A4  88
C128 : 17 A9 34 A2 04 A0 06 20  13
C130 : A4 17 A2 34 20 C9 FF A9  33
C138 : 14 20 D2 FF A9 34 20 C3  55
C140 : FF A9 00 85 FB A9 4F 85  1A
C148 : FC A9 00 8D FC 17 20 21  16
C150 : 12 EE FC 17 AC FC 17 C0  8C
C158 : 32 D0 F3 A9 32 20 C3 FF  58
C160 : A9 33 20 C3 FF 4C E8 CF  C9
C168 : 20 B2 12 D0 0B A2 32 20  4F
C170 : C9 FF A9 0D 4C EE CF 60  81
C178 : A9 00 8D F9 17 A2 32 20  53
C180 : C9 FF 20 90 17 F0 51 A9  F5
C188 : 8D 20 D2 FF 20 B0 17 A2  03
C190 : 33 20 C9 FF A0 00 B9 F0  18
C198 : 17 20 D2 FF C8 C0 08 D0  C8
C1A0 : F5 A9 0D 20 D2 FF A2 32  CD
C1A8 : 20 C9 FF 20 99 12 A9 FE  80
C1B0 : 20 D2 FF EE F9 17 AC F9  16
C1B8 : 17 18 A5 FB 69 08 85 FB  A9
C1C0 : A5 FC 69 00 85 FC C0 50  22
C1C8 : B0 03 4C 36 12 A2 32 20  13
C1D0 : C9 FF A9 0D 20 D2 FF 60  FE
C1D8 : A9 20 20 D2 FF 4C 6C 12  2C
C1E0 : A0 00 8C FA 17 AC FA 17  F4
C1E8 : CC F9 17 D0 01 60 A9 20  8B
C1F0 : 20 D2 FF EE FA 17 4C 9E  2E
C1F8 : 12 A5 FB 85 FD A5 FC 85  98
C200 : FE A9 01 8D FB 17 A2 00  CB
C208 : A0 00 B1 FD C9 00 F0 09  47
C210 : A9 00 8D FB 17 EA EA EA  E6
C218 : EA C8 C0 80 4C 73 17 F0  45
C220 : 03 4C C3 12 18 A5 FD 69  F6
C228 : 80 85 FD A5 FE 69 00 85  E5
C230 : FE E8 E0 05 D0 73 AD FB  D2
C238 : 17 C9 01 60 A2 01 A0 00  35
C240 : B1 FB C9 00 F0 02 A2 00  0B
C248 : C8 C0 08 D0 F3 E0 01 60  97
C250 : 20 BA FF A9 00 20 BD FF  FA
C258 : 20 C0 FF 60 78 A5 01 29  EF
C260 : FD 85 01 A2 00 BA 9D F0  61
C268 : 17 E8 E0 08 D0 F8 A9 80  A9
C270 : 8D F8 17 A0 00 B1 FB A2  16
C278 : 00 0A 90 0B 48 BD F0 17  67
C280 : 0D F8 17 9D F0 17 68 E8  BE
C288 : E0 08 D0 ED 4E F8 17 C8  F9
C290 : C0 08 D0 E1 A5 01 09 02  4F
C298 : 85 01 58 60 EA EA EA EA  47
C2A0 : EA EA EA EA EA EA EA EA  9F
C2A8 : EA 4C C1 12 AA E8 D0 01  A2
```

Listing 7. »Support 802«, fantastische Grafiken auf Ihrem MPS-802-Drucker

Die Statuszeile

Dieses kurze Programm wurde aus der Not geboren, daß bei einem Dateiprogramm in Basic laufende Informationen über den aktuellen Zustand des Programms (aktueller Modus, freier Speicherplatz, Fehlerkanal der Floppy etc.) benötigt wurden.

Das Programm »Statuszeile« (Listing 8) ist leicht von Basic aus zu handhaben und löst das genannte Problem. Es wird mit »SYS 12*4096« gestartet. Listing 9 zeigt, wie man den Text im Speicher ablegen muß.

Programmerklärung (siehe dazu auch Listing 10)

In der »INIT«-Routine werden die Zeiger des Interrupt auf das Programm verbogen. Ab »START« liegt das eigentliche Programm. In der Schleife »LOOP« wird der Text (indirekt adressiert mit x) gelesen und in den Bildschirmspeicher abgelegt. Ab »END« wird mit Hilfe der Betriebssystemroutine »PLOT« die Cursorposition gelesen. Ist der Cursor innerhalb der ersten beiden Zeilen, wird er mit der Routine »SET« in die dritte Zeile gesetzt. »AUS« lenkt den Interrupt auf die normale Interruptroutine im Betriebssystem.

Wird mit SYS 49283 die Routine »STOP« aufgerufen, wird der IRQ-Zeiger wieder auf den ursprünglichen Wert gesetzt und das Programm somit abgeschaltet. Die Hintergrundfarbe der Statuszeilen läßt sich in Speicherstelle $C016 = dezimal 49174 verändern. Der Text, der maximal 80 Zeichen umfassen kann, liegt ab dezimal 49203 und kann dort direkt hineingePOKEt werden. In Listing 9 wird in der Zeile 110 gezeigt, wie man das macht.

Das Assemblerlisting ist ausführlich dokumentiert, um auch Anfängern das Leben leicht zu machen.

Mit anderen Programmen, die den Interrupt beeinflussen, wird dieses Programm ohne Anpassung nicht laufen.

(Uwe Wiards/tr)

```
NAME : STATUS              C000 C090
-------------------------------------
C000 : 78 A9 0D 8D 14 03 A9 C0  C3
C008 : 8D 15 03 58 60 A2 00 BD  82
C010 : 33 C0 9D 00 04 A9 01 9D  D8
C018 : 00 D8 E8 E0 50 F0 03 4C  0C
C020 : 0F C0 38 20 F0 FF E0 02  38
C028 : 10 06 18 A2 02 20 F0 FF  7B
C030 : 4C 31 EA D3 D4 C1 D4 D5  A4
C038 : D3 DA C5 C9 CC C5 CE C5  E5
C040 : C9 CE C2 CC C5 CE C4 D5  4C
C048 : CE C7 A0 C2 D9 A0 D5 D7  24
C050 : C5 A0 D7 C9 C1 D2 C4 D3  02
C058 : A0 A0 A0 C1 C3 C8 D4 DA  34
C060 : C9 C7 A0 DA C5 C9 C3 C8  DB
C068 : C5 CE A0 D4 C5 D8 D4 A0  0F
C070 : A0 A0 A0 D2 C5 D3 D4 A0  72
C078 : C1 D5 C6 C6 D5 C5 CC CC  07
C080 : C5 CE A0 78 A9 31 8D 14  66
C088 : 03 A9 EA 8D 15 03 58 60  58
```

Listing 8. »Status«, Statuszeile im Interrupt

```
10 REM STATUSZEILENEDITOR                          <219>
11 REM BY UWE WIARDS;LEMWERDER                     <192>
12 IF U=1 THEN 15                                  <235>
13 U=1:LOAD"STATUS",8,1                            <153>
15 POKE 53280,PEEK(53281):POKE 650,128             <024>
20 PRINT CHR$(14);"{CLR,3DOWN}STATUSZEILEN
   EDITOR BY UWE WIARDS{3DOWN}"                    <037>
25 PRINT"{2DOWN,SPACE}REST MIT 'SPACE' AUF
   FUELLEN!!"                                      <232>
26 PRINT"{2DOWN,SPACE}STATUSZEILE IST MIT
   SYS49283"                                       <096>
27 PRINT" ABZUSCHALTEN!{3DOWN}"                    <041>
28 PRINT"{2DOWN}CURSORSTEUERUNG NUR MIT 'D
   EL'!!"                                          <106>
30 DIM A(80):FOR I=1 TO 80:PRINT"{RVSON,SP
   ACE,RVOFF}";:NEXT                               <142>
40 PRINT"{2UP}";                                   <015>
45 FOR I=1 TO 80                                   <067>
50 GET A$:IF A$=""THEN 50                          <027>
70 IF A$=CHR$(13)THEN 50                           <236>
80 IF A$=CHR$(20)AND I>1 THEN PRINT"{LEFT,
   RVSON,SPACE,RVOFF,LEFT}";:I=I-1:GOTO 50 <076>
85 IF A$=CHR$(20)THEN 50                           <246>
90 PRINT A$;:A(I)=ASC(A$):NEXT                     <014>
110 FOR I=1 TO 80:POKE 49202+I,A(I):NEXT           <127>
120 SYS 12*4096                                    <253>
```

Listing 9. »Editor«, so erstellt man mit Listing 8 Statuszeilen

```
;statuseinblendung im interupt
;     uwe wiards
;     2874 lemwerder

; assi-fse-assembler 4.12 (c) d.zabel

;2statuszeilen = max 80 zeichen
;ab startadresse + dez 51 text!

*=$c000    ;programmadresse

nzei=2          ;anzahl der zeilen
nlett=40*nzei   ;anz. buchst.
irqv=$0314      ;irq-pointer
plot=$fff0      ;plot   cursor pos l/s
norm=$ea31      ;norm irq
scrb=$0400      ;screenbeginn
farb=$d800      ;farbram beginn


init  sei                     ;irq aus
      lda#<start              ;pointer irqv
      sta irqv                ;auf
      lda#>start              ;start setzen
      sta irqv+1              ;lo u. hi-byte
      cli                     ;irq ein
      rts
start ldx#$00                 ;x initalisieren
loop  lda text,x              ;adr text + x
      sta scrb,x              ;adr scrb + x
      lda#$01                 ;farbe
      sta farb,x              ;setzt farbe
      inx                     ;x=x+1
      cpx#nlett               ;vergleiche
      beq end                 ;0 dann end
      jmp loop
end   sec  ;carry setzen plot liest
      jsr plot
      cpx#nzei
      bpl aus
set   clc  ;carry loe. plot schreibt
      ldx#nzei
      jsr plot
aus   jmp norm
text  .text "STATUSZEILENEINBLENDUNG "
      .text "BY UWE WIARDS    "
      .text "ACHTZIG ZEICHEN TEXT       "
      .text "REST AUFFUELLEN "

stop  sei
      lda#<norm               ;irq-pointer
      sta irqv                ;auf normwert
      lda#>norm               ;zuruecksetzen
      sta irqv+1
      cli
      rts
```

Listing 10. Der dokumentierte Quelltext zu Listing 8

Drei nützliche Befehle

Die Befehlserweiterung »renumber« (Listing 11) stellt Ihnen drei sehr nützliche Befehle zur Verfügung. Als erstes sollten Sie das Programm mit Hilfe des MSE eingeben und speichern. Danach laden Sie es wie ein normales Basic-Programm und starten es mit »RUN«. Der Wortschatz Ihres C 64 umfaßt nun einige neue Kommandos:

»-BYE«	Schaltet diese Befehlserweiterung aus.
»-OLD«	Holt bereits mit »NEW« gelöschte Programme wieder in den Speicher zurück.
»-RENUMBER x,y«	Numeriert das gesamte Programm neu. Alle Zeilennummern hinter (ON..)GOTO,GOSUB,LIST und RUN werden dabei berücksichtigt.
»-UNPACK x,y«	Zieht alle Befehle, soweit dies sinngemäß erlaubt ist, in extra Zeilennummern.
»-PACK x,y,z«	Fügt die einzelnen Befehle eines Programmes so eng wie möglich zusammen.

x,y und z sind hier Variablen für Zahlen zwischen 0 und 63999, wobei x (Defaultwert 0) die Anfangszeilennummer des neuen Programmes, y (Defaultwert 1) den Abstand der einzelnen Zeilennummern und z (Defaultwert 253) die maximale Zeilenlänge in Bytes darstellen.

Es ist so auch möglich, eine kleinere Zeilennummer einer größeren folgen zu lassen (zum Beispiel zum Programmschutz). Man muß hier nur die Anfangszeilennummer oder die Schrittweite groß genug wählen. (Das kommt daher, daß die Summe zweier 16-Bit-Zahlen eine 17-Bit-Zahl ergeben kann, wobei das 17. Bit aus Speicherplatzmangel intern unter den Tisch fallengelassen wird.) Inwieweit dies sinnvoll ist, sei jedem selbst überlassen. Es sei aber vielleicht noch erwähnt, daß weder der Basic-Interpreter noch dieses Programm bei einem Sprungbefehl eine kleinere Zeilennummer hinter einer größeren vermuten (Fehlermeldung).

Die Defaultnummer der maximalen Zeilenlänge (z) wurde aus gutem Grund »nur« auf 253 gesetzt. Bei diesem Wert ist noch ein einwandfreies Laden, Listen und Verbessern möglich.

Wird z größer als 253 gewählt, so können folgende Probleme auftreten:

a) Beim Laden erscheint kein Cursor mehr. Dies tritt auf, weil der Basic-Interpreter nach jedem Laden versucht, das Basic-Programm neu zu »binden«. Falls 256 Byte lang kein 00-Byte auftritt, sucht er sich praktisch zu Tode. Abhilfe schaffen (vor dem Laden) folgende Eingaben:

```
POKE44,191
LOAD"name",8,1
POKE44,8
```

b) Ein List-Befehl listet nicht mehr das gesamte Programm. Falls eine Zeile mit mehr als 253 Zeichen auftritt, so werden nur diese aufgelistet und dann abgebrochen.

Es ist jetzt auch nicht mehr möglich, das Programm zu verändern (Programmierschutz).

Eine dritte Schwierigkeit wird vom Programm selbständig behoben. Ein Sprungbefehl über 253 Nicht-Null-Bytes einer Zeile ist nicht möglich.

Das Programm akzeptiert auch offengelassene Anführungsstriche und schließt sie bei einem Pack-Befehl selbständig.

Beispielsweise wird das folgende Basicprogramm

```
5  PRINTCHR$(147);:B$=""
10 IFLEN(B$)<255 THENGETA$:B$=B$+A$
15 PRINTB$:POKE53281,PEEK(53281)-1:GOTO10
```

durch »-UNPACK« zu

```
0 PRINTCHR$(147);
1 B$=""
2 IFLEN(B$)<255THENGETA$:B$=B$+A$
3 PRINTB$
4 POKE53281,PEEK(53281)-1
5 GOTO2
```

und durch »-PACK 5,5« wieder zu ersterem.

Daß das Programm arbeitet, sieht man an dem blinkenden Stern am Bildschirmrand rechts oben.

Bei einem »PACK«-Befehl ändert der Stern mehrmals die Farbe, da hier zusätzlich ein »UNPACK«- und ein »RENUMBER«-Befehl durchgeführt wird.

Fehlermeldungen

Während der Arbeit gibt das Programm Fehlermeldungen in folgendem Format aus:

xx yyyyy zzzzzzz,	wobei
xx	eine interne Fehlernummer,
yyyyy	die neue Fehlernummer, in welcher der Fehler auftritt
zzzzzzz	die Fehlermeldung selber ist.

Es gibt folgende Fehlermeldungen:

Syntax error	Eventuelle Variablen als Sprungadressen können nicht geändert werden.
No such line	Sprungbefehl zu einer nicht existierenden Zeile
Illegal line number	Zeilennummer zu groß

Zeilennummern, die nicht gefunden werden, werden bei den letzten beiden Fehlern einheitlich in 64000 geändert. Hier besteht keine »Verwechslungsgefahr«, da in Basic keine Zeilennummern ab 63999 eingegeben werden können.

Sonstiges

Das Programm benötigt nur den Speicherbereich von $C000 bis $CFFF. Zu eventuellen Programmiererweiterungen verhält sich »renumber« tolerant, sofern sich nicht die Speicherbereiche überlappen. Hierzu sollte »renumber« als letzteres geladen werden. Zweimaliges Initialisieren führt zum Absturz.

(Michael Rothmeier/tr)

```
NAME : RENUMBER          0801 13EB
-----------------------------------
0801 : 42 08 00 00 9E 32 31 31   EA
0809 : 36 3A A2 3A 8F 22 0D 91   AE
0811 : 12 D2 45 4E 55 4D 42 45   FB
0819 : 52 20 36 34 20 20 20 20   53
0821 : 20 20 31 39 38 36 92 0D   5E
0829 : 0D 0E 28 C3 29 20 CD 49   1D
0831 : 43 48 41 45 4C 20 D2 4F   41
0839 : 54 48 4D 45 49 45 52 0D   CF
0841 : 00 00 00 A9 89 85 58 A9   F0
0849 : CB 85 59 A9 EB 85 5A A9   0A
0851 : 13 85 5B A9 62 85 5F A9   56
0859 : 08 85 60 20 BF A3 4C 85   95
0861 : C0 5B C1 6D C1 84 C1 8D   4F
0869 : C1 A0 C1 00 00 00 00 00   EB
0871 : 00 42 4F 50 52 55 00 00   40
0879 : 00 8D 89 9B 8A A7 00 3A   70
0881 : 8B 8F 00 CF 4B D3 59 4E   23
0889 : 54 41 58 20 45 52 52 4F   67
0891 : 52 C9 4C 4C 45 47 41 4C   91
0899 : 20 4C 49 4E 45 20 4E 55   35
08A1 : 4D 42 45 52 CE 4F 20 53   39
08A9 : 55 43 48 20 4C 49 4E 45   89
08B1 : A0 20 00 00 14 14 0D 0D   92
08B9 : 54 4F 54 41 4C 20 45 52   72
08C1 : 52 4F 52 53 3A 20 0D 0D   AD
08C9 : 4F 4C 44 00 4E 45 57 00   BC
08D1 : 20 50 52 4F 47 52 41 4D   3E
08D9 : 20 3A 20 20 42 59 54 45   ED
08E1 : 53 0D 01 0F 09 00 A2 10   18
08E9 : BD 79 CB 20 D2 FF CA 10   D2
08F1 : F7 AD 08 03 8D B7 C0 AD   16
08F9 : 09 03 8D B8 C0 A9 A7 8D   11
0901 : 08 03 A9 C0 8D 09 03 60   FB
0909 : 20 73 00 C9 AB F0 0B A5   D6
0911 : 7A D0 02 C6 7B C6 7A 4C   BD
0919 : AE A7 20 73 00 85 D7 A0   DE
0921 : 00 84 34 88 CB B9 10 C0   9E
0929 : D0 05 A2 0B 4C 3A A4 C5   3B
0931 : D7 D0 F1 8C FB C0 0E F8   3E
0939 : C0 20 DC C6 A2 00 A1 7A   BF
0941 : F0 04 C9 3A D0 2A 78 20   6D
0949 : AA C1 20 F0 C6 A9 93 20   42
0951 : D2 FF A9 C0 48 A9 F9 48   EF
0959 : 6C 00 C0 A9 20 8D 27 04   3E
0961 : 20 A7 CA 20 B5 C1 58 20   16
0969 : 79 00 20 63 A6 4C E7 A7   13
0971 : 20 63 C3 A6 0F F0 05 A2   DA
0979 : 0E 4C 3A A4 A5 0D 85 7A   9E
0981 : A5 0E 85 7B A5 07 D0 08   E4
0989 : A1 0D F0 BA C9 3A F0 B6   E4
0991 : A6 07 D0 0E A2 00 A1 7A   56
0999 : C9 2C F0 10 A5 34 F0 9C   AF
09A1 : D0 88 A6 34 A5 10 95 22   5B
09A9 : A5 11 95 23 E6 34 E6 34   B5
09B1 : A6 34 E0 0B D0 86 4C C9   B0
09B9 : C0 6C 00 C0 AD B7 C0 8D   7E
09C1 : 08 03 AD B8 C0 8D 09 03   70
09C9 : A0 6A 20 2F F1 60 A0 01   33
09D1 : 98 91 2B 20 33 C6 A5 2F   5F
09D9 : 69 02 8D BD CB A5 30 69   DC
09E1 : 00 8D BE CB 60 E6 17 20   AB
09E9 : E4 C7 20 8D C1 60 20 46   97
```

Listing 11. »Renumber«, Befehlserweiterung mit drei sehr nützlichen Befehlen

```
09F1 : CB 20 3E C7 20 7A C5 20  82
09F9 : 76 C6 20 88 C5 20 33 C6  A3
0A01 : 60 20 46 CB 20 5D C7 20  C9
0A09 : 8D C1 60 A2 40 B5 00 9D  D0
0A11 : 90 CB CA D0 F8 60 A2 40  F1
0A19 : BD 90 CB 95 00 CA D0 F8  4F
0A21 : 60 A9 00 85 05 F0 40 AE  3D
0A29 : BC CB 86 7B AE BB CB D0  75
0A31 : 02 C6 7B CA 86 7A A2 00  95
0A39 : 86 09 86 05 86 08 86 12  6D
0A41 : A0 01 B1 7A D0 08 C8 B1  F1
0A49 : 7A D0 03 E6 05 60 A0 03  A5
0A51 : B1 7A 85 0B C8 B1 7A 85  11
0A59 : 0C 18 A5 7A 69 05 85 7A  F4
0A61 : 90 0B E6 7B 4C 0C C2 E6  9E
0A69 : 7A D0 02 E6 7B A2 00 A1  B9
0A71 : 7A F0 C3 20 0F C7 B0 EF  2A
0A79 : 85 0A A0 00 B9 18 C0 F0  6D
0A81 : E6 C8 C5 0A D0 F6 60 AD  20
0A89 : BB CB 85 02 AD BC CB 85  C6
0A91 : 03 A5 02 85 1F A5 03 85  CE
0A99 : 20 AD 19 C5 48 AD 1A C5  75
0AA1 : 48 A9 F0 8D 19 C5 A9 C2  98
0AA9 : 8D 1A C5 A0 01 84 19 84  6A
0AB1 : 06 88 84 04 A0 00 B1 02  72
0AB9 : D0 09 C8 B1 02 D0 03 4C  C1
0AC1 : E3 C2 88 84 09 84 08 84  96
0AC9 : 12 A6 04 A0 02 B1 02 9D  34
0AD1 : 00 CC C8 B1 02 9D 00 CD  48
0AD9 : 88 A5 22 9D 00 CE C8 A5  55
0AE1 : 23 9D 00 CF A5 21 D0 0A  88
0AE9 : 88 A5 22 91 02 C8 A5 23  42
0AF1 : 91 02 C8 D0 02 E6 03 B1  96
0AF9 : 02 F0 29 85 0A A6 21 D0  6A
0B01 : F1 20 0F C7 B0 EC A2 FF  BC
0B09 : E8 BD 18 C0 F0 E4 C5 0A  4F
0B11 : D0 F6 18 98 65 02 85 02  F6
0B19 : 90 02 E6 03 20 18 C5 A0  DF
0B21 : 00 4C 91 C2 18 A5 22 65  06
0B29 : 24 85 22 A5 23 65 25 85  4A
0B31 : 23 38 98 65 02 85 02 90  B9
0B39 : 02 E6 03 E6 04 F0 03 4C  B8
0B41 : 53 C2 C6 19 68 8D 1A C5  B1
0B49 : 68 8D 19 C5 A9 00 85 06  34
0B51 : 60 A5 0A C9 A7 D0 0A A0  AA
0B59 : 01 B1 02 F0 04 C9 01 D0  06
0B61 : 0B 18 A5 02 69 03 85 02  EB
0B69 : 90 02 E6 03 60 20 48 CB  D4
0B71 : A0 00 18 A5 2F E5 31 85  BE
0B79 : 35 A5 30 E5 32 85 36 A5  BD
0B81 : 30 C5 34 90 1C D0 06 A5  5F
0B89 : 2F C5 33 90 14 B1 2F 91  28
0B91 : 33 C8 D0 04 E6 30 E6 34  D1
0B99 : E6 35 D0 F1 E6 36 D0 ED  CB
0BA1 : 60 18 A5 33 E5 35 85 33  62
0BA9 : A5 34 E5 36 85 34 B1 31  CC
0BB1 : 91 33 88 C0 FF D0 04 C6  3A
0BB9 : 32 C6 34 E6 35 D0 EF E6  A0
0BC1 : 36 D0 EB 60 A5 7A 85 0D  C5
0BC9 : A5 7B 85 0E A0 00 84 10  8B
0BD1 : 84 11 84 0F 84 07 C8 D0  26
0BD9 : 02 E6 0E B1 0D 29 7F C9  B4
0BE1 : 20 F0 F3 B1 0D 38 E9 30  47
0BE9 : 90 3D C9 0A B0 39 E6 07  4A
0BF1 : AA A5 10 85 31 A5 11 85  B2
0BF9 : 32 C9 19 B0 24 06 10 26  6B
0C01 : 11 06 10 26 11 A5 10 65  28
0C09 : 31 85 10 A5 11 65 32 85  C6
0C11 : 11 06 10 26 11 18 8A 65  B5
0C19 : 10 85 10 90 B9 E6 11 D0  BB
0C21 : B5 A9 02 85 0F D0 AF 18  42
0C29 : 98 65 0D 85 0D 90 02 E6  93
0C31 : 0E A5 0F F0 08 A9 FA 85  B8
0C39 : 11 A9 02 85 10 60 A5 11  0D
0C41 : C9 FA 90 0A A5 10 85 0F  FC
0C49 : 20 32 CA 4C 26 C4 A5 18  0E
0C51 : D0 3D A5 17 D0 09 A9 03  0E
0C59 : 85 0F 20 32 CA D0 28 A5  D3
0C61 : 25 D0 14 A6 24 CA D0 0F  C2
0C69 : A5 23 69 80 C9 FA 90 02  C5
0C71 : A9 00 85 11 4C 2E C4 A6  34
0C79 : 23 86 11 A6 22 E8 86 10  9C
0C81 : D0 0D E6 11 4C 2E C4 A9  50
0C89 : FA 85 11 A9 00 85 10 A9  7F
0C91 : 04 85 32 A9 00 85 2F 85  0E
0C99 : 30 85 31 A2 11 CA F0 18  88
0CA1 : 06 10 26 11 26 2F A5 2F  2C
0CA9 : C9 0A 26 30 26 31 C9 0A  2E
0CB1 : 90 EB E9 0A 85 2F B0 E5  53
0CB9 : 18 69 30 A6 32 95 1A C6  2C
0CC1 : 32 30 0B A5 30 85 10 A5  3E
0CC9 : 31 85 11 4C 32 C4 60 A5  A1
0CD1 : 06 D0 13 A2 00 E6 7A D0  1B
0CD9 : 02 E6 7B A1 7A 29 7F C9  E4
0CE1 : 20 F0 F2 A1 7A 60 A2 00  9F
0CE9 : E6 02 D0 02 E6 03 A1 02  56
0CF1 : 29 7F C9 20 F0 F2 A1 02  81
0CF9 : 60 20 63 C3 A5 0A C9 A7  DC
0D01 : D0 0D A4 07 D0 09 C8 B1  3E
0D09 : 7A D0 66 A9 FA 85 11 A5  25
0D11 : 0D 85 2F A5 0E 85 30 AC  89
0D19 : BD CB 84 31 AC BE CB 84  FC
0D21 : 32 18 A5 7A 85 37 69 03  D6
0D29 : 85 7A 85 33 A5 7B 85 38  70
0D31 : 69 00 85 7B 85 34 E6 33  67
0D39 : D0 02 E6 34 38 AD BD CB  CA
0D41 : E5 2F 8D BD CB AD BE CB  96
0D49 : E5 30 8D BE CB 18 AD BD  31
0D51 : CB 65 33 8D BD CB AD BE  BC
0D59 : CB 65 34 8D BE CB 20 0C  78
0D61 : C3 A0 01 A9 00 91 37 C8  E5
0D69 : A5 10 91 37 C8 A5 11 91  83
0D71 : 37 60 20 C6 C1 A5 05 D0  B8
0D79 : 61 20 99 C6 A5 0A C9 89  0E
0D81 : F0 04 C9 8D D0 0D A5 08  B3
0D89 : F0 24 20 6E C4 C9 2C F0  8E
0D91 : E8 D0 1E C9 9B D0 0D 20  57
0D99 : 6E C4 C9 AB D0 13 A9 8A  B2
0DA1 : 85 0A D0 D5 C9 A7 D0 06  43
0DA9 : 20 6E C4 4C 5F C5 20 6E  3C
0DB1 : C4 C9 00 F0 0B C9 3A F0  42
0DB9 : 07 A0 01 84 0F 20 32 CA  32
0DC1 : A5 06 D0 0E A5 7A D0 02  D5
0DC9 : C6 7B C6 7A 20 C0 C1 4C  F5
0DD1 : 14 C5 A5 02 D0 02 C6 03  B0
0DD9 : C6 02 60 A9 98 8D 19 C5  D3
0DE1 : A9 C4 8D 1A C5 20 11 C5  C0
0DE9 : 60 A9 98 8D 19 C5 A9 C5  E8
0DF1 : 8D 1A C5 C6 17 20 11 C5  18
0DF9 : 60 A5 0A C9 A7 D0 0B A0  56
0E01 : 01 B1 7A F0 05 C9 01 F0  1C
0E09 : 01 60 A0 01 B1 7A 85 18  B8
0E11 : C8 B1 7A 85 10 C8 B1 7A  04
0E19 : 85 11 20 DD C3 A0 FF C8  BD
0E21 : C0 04 F0 07 B9 1A 00 C9  00
0E29 : 30 F0 F4 84 37 18 A5 7A  5F
0E31 : 69 04 85 2F A5 7B 69 00  BF
0E39 : 85 30 AE BE CB 86 32 AE  71
0E41 : BD CB D0 02 C6 32 CA 86  8E
0E49 : 31 A5 7B 85 39 38 A5 7A  BD
0E51 : 85 38 E5 37 B0 02 C6 7B  80
0E59 : 18 69 05 85 7A 90 02 E6  1A
0E61 : 7B 85 33 A6 7B E6 33 D0  9E
0E69 : 01 E8 86 34 38 AD BD CB  86
0E71 : E5 37 B0 03 CE BE CB 18  C1
0E79 : 69 02 8D BD CB [illegible] 03 EE  29
0E81 : BE CB 20 0C C3 A0 37 A0  3E
0E89 : 01 B5 1A 91 38 C8 E8 E0  4D
0E91 : 05 D0 F6 60 AD BB CB 85  BB
0E99 : 2F 85 31 AD BC CB 85 30  2D
0EA1 : 85 32 A0 00 B1 2F D0 05  49
0EA9 : C8 B1 2F F0 29 A0 04 B1  3F
0EB1 : 2F D0 1B 38 98 65 2F 85  93
0EB9 : 2F 90 02 E6 30 A0 00 91  B9
0EC1 : 31 AA C8 A5 30 91 31 86  8F
0EC9 : 31 85 32 4C 41 C6 C8 D0  E2
0ED1 : DE E6 30 4C 4E C6 60 A9  A8
0ED9 : 99 8D 19 C5 A9 C6 8D 1A  73
0EE1 : C5 20 26 C2 A5 19 F0 04  87
0EE9 : A5 04 F0 0D 20 11 C5 A5  5B
0EF1 : 19 D0 06 20 30 C2 4C 83  49
0EF9 : C6 60 A5 0A C9 A7 D0 0A  CB
0F01 : A0 01 B1 7A F0 04 C9 01  36
0F09 : D0 32 A0 01 B1 7A C9 01  53
0F11 : F0 1F A0 02 B1 7A 85 2F  5D
0F19 : C8 B1 7A 85 30 20 61 CB  2A
0F21 : D0 0F BD 00 CF 91 7A 88  6D
0F29 : BD 00 CE 91 7A 88 98 91  3E
0F31 : 7A 18 A5 7A 69 03 85 7A  2A
0F39 : 90 02 E6 7B 60 A2 FA 86  07
0F41 : 26 A2 00 86 27 A2 00 86  1E
0F49 : 23 86 22 86 25 E8 86 24  05
0F51 : 60 A2 FE 8E 20 D0 8E 21  99
0F59 : D0 E8 86 28 E8 86 21 86  98
0F61 : 06 86 13 86 14 E8 86 17  11
0F69 : A2 40 86 15 E8 86 16 60  4B
0F71 : C9 22 D0 09 A5 09 49 01  6A
0F79 : 85 09 4C 3A C7 A6 09 D0  55
0F81 : 1C A6 12 D0 18 C9 3A D0  E9
0F89 : 04 A2 00 86 08 C9 91 D0  66
0F91 : 04 A2 01 86 08 C9 8F D0  A6
0F99 : 02 E6 12 18 60 38 60 A2  25
0FA1 : 18 A0 C0 4C 50 C7 A2 1E  CD
0FA9 : A0 C0 4C 50 C7 A2 1F A0  16
0FB1 : C0 8E 1C C2 8C 1D C2 8E  F2
0FB9 : A9 C2 8C AA C2 60 20 45  76
0FC1 : C7 20 C6 C1 A5 05 D0 7C  41
0FC9 : A5 0A C9 3A D0 51 A0 01  49
0FD1 : B1 7A F0 6A C9 3A F0 66  48
0FD9 : 88 98 91 7A A5 7A 85 2F  04
0FE1 : 85 37 A5 7B 85 30 85 34  33
0FE9 : 85 38 18 AD BD CB 85 31  F9
0FF1 : 69 04 8D BD CB AD BE CB  34
0FF9 : 85 32 90 04 EE BE CB 18  80
1001 : A5 7A 69 04 85 7A 85 33  67
1009 : 90 04 E6 7B E6 34 20 0C  6D
1011 : C3 A0 03 A5 0B 91 37 C8  46
1019 : A5 0C 91 37 4C DD C7 A0  24
1021 : 01 B1 7A F0 08 C8 D0 F9  B5
1029 : E6 7B 4C C0 C7 88 C0 FF  BC
1031 : D0 02 C6 7B 18 98 65 7A  F4
1039 : 85 7A 90 02 E6 7B 20 C0  AC
1041 : C1 4C 63 C7 60 A5 22 48  46
1049 : A5 23 48 A5 24 48 A5 25  AC
1051 : 48 A5 26 D0 06 A5 27 D0  DB
1059 : 02 C6 27 E6 26 D0 02 E6  24
1061 : 27 20 A0 C1 20 46 CB 20  9C
1069 : E4 C6 E6 21 A9 00 85 24  87
1071 : 20 3E C7 20 26 C2 A5 19  E7
1079 : F0 07 A5 04 D0 03 4C ED  09
1081 : C9 20 3E C7 A9 FC 8D 19  CE
1089 : C5 A9 C9 8D 1A C5 20 11  B9
1091 : C5 20 4C C7 A9 1F 8D 19  6E
1099 : C5 A9 CA 8D 1A C5 20 11  0A
10A1 : C5 A6 27 E0 FF D0 03 4C  CA
10A9 : D1 C8 A5 1F 85 7A A5 20  2F
10B1 : 85 7B A2 00 A5 2F 85 31  E9
10B9 : A5 30 85 32 18 A5 7A 69  89
10C1 : 03 85 7A 90 02 E6 7B A0  BE
10C9 : 00 84 2F 84 30 A0 00 84  79
10D1 : 09 C8 D0 06 E6 7B E6 30  79
10D9 : E6 32 B1 7A F0 0C C9 2A  7F
10E1 : D0 EF A5 09 49 01 85 09  F8
10E9 : 10 E7 38 98 65 7A 85 7A  43
10F1 : 90 02 E6 7B 18 A5 09 69  51
10F9 : FF 98 65 2F 85 2F 90 02  9C
1101 : E6 30 18 A5 09 69 FF 98  C7
1109 : 65 31 85 31 90 02 E6 32  A7
1111 : BD 00 CE D0 11 A5 27 C5  02
1119 : 32 90 08 D0 10 A5 26 C5  02
1121 : 31 B0 0A FE 00 CE E8 E4  F1
1129 : 04 D0 89 F0 05 E8 E4 04  49
1131 : D0 8A A5 27 D0 06 A5 26  B5
1139 : C9 FC 90 74 38 A5 1F E9  34
1141 : 02 85 7A A5 20 E9 00 85  B5
1149 : 7B 20 3E C7 20 C0 C1 A5  B7
1151 : 05 D0 5D A5 0B 85 2F A5  AF
1159 : 0C 85 30 20 61 CB D0 50  90
1161 : A0 00 84 2F A0 00 84 09  37
1169 : C8 F0 3D B1 7A F0 0C C9  22
1171 : 22 D0 F5 A5 09 49 01 85  17
1179 : 09 10 ED 18 98 65 7A 90  C8
1181 : 03 E6 7B 18 69 04 85 7A  9B
1189 : 90 02 E6 7B 18 A5 09 69  E9
1191 : FF 98 65 2F 85 2F B0 10  D0
1199 : C9 FC B0 0C BD 01 CE D0  4F
11A1 : AB E8 E4 04 F0 0A D0 BC  96
11A9 : FE 00 CE E8 E4 04 D0 9C  63
11B1 : A2 00 18 A5 1F 85 02 A5  80
11B9 : 20 85 03 90 06 E6 02 D0  B0
11C1 : 02 E6 03 A5 1F 69 03 85  00
11C9 : 1F 90 02 E6 20 A0 00 84  9E
11D1 : 09 C8 D0 02 E6 20 B1 1F  27
11D9 : F0 09 C9 22 D0 F3 E6 09  5F
11E1 : 4C 70 C9 18 98 65 1F 85  17
11E9 : 1F 90 02 E6 20 E8 E4 04  93
11F1 : F0 52 BD 00 CE 38 D0 BB  E3
11F9 : 18 A5 09 29 01 85 09 D0  4D
1201 : 01 38 A5 1F 85 33 69 03  09
1209 : 85 2F A5 20 85 34 69 00  33
1211 : 85 30 38 A5 09 D0 01 18  BC
1219 : AD BD CB 85 31 E9 03 8D  D2
1221 : BD CB AD BE CB 85 32 E9  8C
1229 : 00 8D BE CB 8A 48 20 0C  9C
1231 : C3 68 AA A0 00 A5 09 F0  1A
1239 : 05 A9 22 91 1F C8 A9 3A  21
1241 : 91 1F D0 89 A5 19 D0 06  3A
1249 : 20 30 C2 4C 15 C8 68 85  00
1251 : 25 68 85 24 68 85 23 68  A0
1259 : 85 22 C6 21 60 20 63 C3  E1
1261 : A5 10 85 2F A5 11 85 30  AF
1269 : 20 61 CB D0 05 A9 01 9D  24
1271 : 00 CE A6 0E 86 7B A6 0D  3D
1279 : D0 02 C6 7B CA 86 7A 60  F7
1281 : A5 0B 85 2F A5 0C 85 30  24
1289 : 20 61 CB D0 05 A9 01 9D  44
1291 : 01 CE 60 48 A5 17 D0 6E  4E
1299 : E6 13 D0 02 E6 14 E6 15  52
12A1 : A5 15 C9 5B D0 06 A9 41  15
12A9 : 85 15 E6 16 A9 0D 20 D2  5E
12B1 : FF A9 01 20 D2 FF A5 15  B7
12B9 : 20 D2 FF A5 16 20 D2 FF  A5
12C1 : A5 0B 85 10 A5 0C 85 11  42
12C9 : 20 2E C4 A9 3A 20 D2 FF  57
12D1 : A9 20 20 D2 FF A2 00 B5  6D
12D9 : 1A 20 D2 FF E8 E0 05 D0  03
12E1 : F6 A9 20 20 D2 FF A4 0F  96
12E9 : A2 FF C8 88 30 08 E8 BD  30
12F1 : 22 C0 10 FA 30 F5 29 7F  2D
12F9 : 20 D2 FF E8 BD 22 C0 10  AF
1301 : F7 A9 00 20 D2 FF 68 60  60
```

Listing 11. »Renumbers« (Fortsetzung). Bitte mit dem MSE eingeben.

```
1309 : A2 00 BD 4F C0 20 D2 FF  5D
1311 : E8 E0 16 D0 F5 A5 13 85  ED
1319 : 10 A5 14 85 11 20 24 CB  EC
1321 : A2 00 A0 01 BD 65 C0 F0  F7
1329 : 06 20 D2 FF E8 D0 F5 86  EE
1331 : 33 A2 00 BD 6F C0 20 D2  90
1339 : FF E8 E0 0B D0 F5 38 98  15
1341 : F0 11 A5 2D ED BB CB 85  C0
1349 : 10 A5 2E ED BC CB 85 11  D7
1351 : 98 D0 10 AD BD CB ED BB  74
1359 : CB 85 10 AD BE CB ED BC  1C
1361 : CB 85 11 20 24 CB A2 00  62
1369 : BD 7A C0 20 D2 FF E8 E0  2A
1371 : 07 D0 F5 A6 33 E8 88 10  EF
1379 : AB A9 00 20 D2 FF 20 D2  50
1381 : FF 20 D2 FF 60 20 2E C4  8E
1389 : A2 FF E8 B5 1A E0 04 F0  B6
1391 : 0B C9 30 D0 07 A9 20 20  25
1399 : D2 FF D0 EE 20 D2 FF E8  E7
13A1 : B5 1A E0 05 D0 F6 60 E6  50
13A9 : 28 AD 27 04 C9 2A D0 04  2B
13B1 : A9 20 D0 02 A9 2A 8D 27  4F
13B9 : 04 A4 28 B9 81 C0 8D 27  F3
13C1 : D8 60 A2 00 BD 00 CC C5  0D
13C9 : 2F D0 07 BD 00 CD C5 30  C0
13D1 : F0 07 E8 E4 04 D0 ED A2  DF
13D9 : 01 60 11 34 36 20 52 45  0D
13E1 : 42 4D 55 4E 45 52 12 11  3A
13E9 : 05 93 00 00 00 00 00 00  B8
```

Listing 11. »Renumber« (Schluß)

Die etwas andere Dateiverwaltung

»TAXI-SCHOOL« (Listing 12) ist ein System, welches die Erstellung von Dateien mit maximal 512 Zeilen x 24 Zeichen erlaubt. Das Besondere an diesem System ist die Möglichkeit, die einzelnen Zeilen auf bestimmte Weise zu verknüpfen.

1. Starten des Programms
- Laden Sie das Programm mit »LOAD "TAXI-SCHOOL",8«
- Starten Sie das Programm mit »RUN«
- Das Programm verwendet einen anderen Zeichensatz (Listing 13), als ihn der C 64 zur Verfügung stellt. Dieser Zeichensatz wird automatisch geladen. Er muß auf der Diskette unter dem Namen »SIGNS« gespeichert sein. Er beinhaltet einige Sonderzeichen, die zusammen mit der Commodore-Taste aufgerufen werden können.
- Wollen Sie einen eigenen Zeichensatz verwenden, so muß dieser unter dem Namen »SIGNS« auf Diskette gespeichert sein und von $C000 bis $C800 gehen (also nur ein einfacher Zeichensatz von 256 Zeichen).

2. Editieren von Dateien
- Es stehen 512 Zeilen x 24 Zeichen zur Verfügung.
- Eine Zeile wird durch <RETURN> in den Speicher übernommen.
- <HOME> setzt den Cursor an den Anfang der Zeile.
- <CLR> löscht die Cursorzeile (der Speicher bleibt dadurch unberührt.
- <CTRL @> löscht den gesamten Speicher.
- Die Cursortasten haben ihre übliche Funktion.

Funktionstasten:

F2: insert
- Alles ab Cursorzeile wird um eine Zeile hinuntergeschoben. Die Cursorzeile wird dadurch für einen Eintrag frei.
- Diesen Befehl sollten Sie nicht mehr verwenden, wenn Sie die »BASIS-STREET«-Beziehungen bereits festgelegt haben (siehe dazu etwas weiter unten).

F4: delete
- Die Cursorzeile wird gelöscht. Alle übrigen Einträge rücken auf.
- Für diesen Befehl gilt das gleiche wie für den Insert-Befehl.

3. Laden und Speichern von Dateien

F6: load
- Die Zeichen links vom Cursor werden als Filename interpretiert und der Befehl »LOAD "(FILENAME)",8,1« ausgeführt.
- Wurde ein $-Zeichen eingegeben, so erscheint das Inhaltsverzeichnis der Diskette.

F8: save
- Die Zeichen links vom Cursor werden als Filename interpretiert und der Befehl »SAVE "(FILENAME)",8,1« ausgeführt.
- Es wird von der ersten bis zur Zeile über dem Cursor (einschließlich) unter dem angegebenen Filenamen auf der Diskette gespeichert.

4. Die BASIS-STREET-Verbindungen

Das Programm gibt Ihnen die Möglichkeit, zu jeder der 512 Zeilen 20 Zeilen als Unterzeilen zu bestimmen. Eine Unterzeile bezeichnen wir mit »STREET«. Jede Zeile ist »BASIS«, aber nur die auserwählten Zeilen sind »STREETS«.

Wollen Sie eine Zeile als »STREET« kennzeichnen, so muß vorher festgelegt worden sein, welche Zeile die »BASIS« ist.

F3: basis
- Bringen Sie den Cursor in die Zeile, die die »BASIS« sein soll.
- Drücken Sie <F3>.
- Bis zum nächsten <F3> liegt die »BASIS« nun fest.

F5: street
- Bringen Sie den Cursor in die Zeile, die Sie als »STREET« kennzeichnen wollen.
- Drücken Sie <F5>.
- Die Zeile wird als »STREET« zur aktuellen »BASIS« gekennzeichnet.

Wollen Sie zu einer beliebigen Zeile die dazugehörigen »STREETS« auflisten lassen, so benutzen Sie dazu den Drive-Befehl <F7>. Dazu muß in der Cursorzeile der Inhalt der Zeile stehen, zu der Sie die »STREETS« auflisten wollen. Dies können Sie erreichen, indem Sie den Cursor in die entsprechende Zeile bringen oder den Text in die aktuelle Zeile schreiben.

F7: drive
- Es wird die ausgewählte »BASIS« am oberen Feldrand ausgegeben und darunter mit zwei Zeilen Abstand die dazugehörigen »STREETS«.
- Durch diesen Befehl werden die Funktionen <F3> und <F5> sowie das Übernehmen einer Zeile in den Speicher durch <RETURN> ausgeschaltet. Das heißt, daß bis zum nächsten <F1> oder Scrollen keine »BASIS« oder »STREET« gekennzeichnet werden kann und auch keine Zeile in den Speicher übernommen wird.
- Bewegen Sie den Cursor aus dem Feld oder rufen Sie den Search-Befehl auf, so wird zurück in die Liste gesprungen. Jetzt sind alle Befehle wieder aktiv.

Zum Auffinden einer bestimmten Zeile dient der Search-Befehl <F1>.

F1: search
- Es wird ein CLR ausgeführt.
- Beim nächsten <RETURN> wird nach einer Zeile gesucht, die mit der gleichen Zeilenfolge beginnt wie die aktuelle Cursorzeile.
- Zeichen, die aus dem Vergleich ausgeschlossen werden sollen, können durch ein »?« ersetzt werden.

Anwendungsbeispiel Taxischule

Beginnen Sie damit, die Straßennamen Ihrer Stadt zu speichern. Ordnen Sie die Straßen zum Beispiel nach Ortsteilen. In jeder Zeile steht nun ein Straßenname. Kennzeichnen Sie nun nacheinander die einzelnen Straßen als »BASIS« und die davon abzweigenden Straßen als »STREETS«.

Einige Zeilen können auch Informationen enthalten wie »Einbahnstraße« oder »Kirche« oder ähnliches. Soll einer Straße eine Information zugeordnet werden, so wird die Straße als »BASIS« gekennzeichnet und die Information als »STREET«.

Es kann auch der Name einer anderen Datei als »STREET« zu einer »BASIS« dienen. Sie erhalten beim Auflisten der »STREETS« dann zum Beispiel die Information, daß die Straße in einem anderen Ortsteil weitergeht.

Haben Sie die Dateien nach Ihren Vorstellungen erstellt und gespeichert, können Sie Ihre Lernfortschritte mit diesem Programm testen. Dazu wählen Sie zunächst einen Ausgangs- und einen Endpunkt.

Bringen Sie dann den Cursor in die Zeile, in der der Ausgangspunkt steht. Drücken Sie nun <F7>. Es wird der Ausgangspunkt angezeigt und alle davon aus erreichbaren Straßen. Wählen Sie eine aus und drücken Sie wieder <F7>. Dies wiederholen Sie so lange, bis Sie Ihr gewähltes Ziel erreicht haben. Sollte letzteres nicht gelingen, so haben Sie den Stadtplan noch nicht richtig im Kopf und müssen noch lernen.

Für diese oder ähnliche Anwendungen wurde das Programm »TAXI-SCHOOL« entwickelt. Da es sich im Grunde jedoch um ein kleines Datenbank-System handelt, können Sie natürlich auch andere Probleme damit bewältigen (Notizbuch, Wörterbuch und so weiter).

(Christoph Gladenbeck/tr)

```
NAME : TAXI-SCHOOL       0801 101A
-----------------------------------
0801 : 0C 08 01 00 9E 20 32 30   66
0809 : 36 32 00 00 00 20 EA 0D   1F
0811 : A9 30 8D 18 D0 A9 94 8D   00
0819 : 00 DD A9 CC 8D 88 02 A9   84
0821 : 8E 20 D2 FF A9 08 20 D2   75
0829 : FF A9 00 8D 05 0B 85 FB   65
0831 : A9 20 85 FC 8D 06 0B A0   62
0839 : 3F B9 64 0D 91 FB 88 C0   AC
0841 : FF D0 F6 A5 FB 18 69 40   C1
0849 : 85 FB A5 FC 69 00 85 FC   7B
0851 : C9 A0 D0 E3 A9 00 8D 20   2C
0859 : D0 8D 21 D0 20 36 0E A9   91
0861 : 1E 20 D2 FF A9 13 20 D2   9D
0869 : FF A9 18 A0 17 A2 20 20   9E
0871 : 05 0A 4C 76 08 18 65 D3   DC
0879 : 8D A9 0A 88 8C A8 0A A4   EE
0881 : D3 8C AA 0A A9 00 8D A7   A6
0889 : 0A 20 AC 0A A5 C6 85 CC   50
0891 : 8D 92 02 F0 F7 78 A5 CF   7F
0899 : F0 0C A5 CE AE 87 02 A0   43
08A1 : 00 84 CF 20 13 EA 20 B4   4E
08A9 : E5 20 B0 08 4C 8D 08 C9   B0
08B1 : 0D D0 0B 20 82 0B AD AA   7A
08B9 : 0A 85 D3 4C 98 09 C9 8D   18
08C1 : F0 F1 C9 13 D0 06 AD AA   C8
08C9 : 0A 85 D3 60 C9 93 D0 10   33
08D1 : AC AA 0A A9 20 91 D1 C8   F1
08D9 : CC A9 0A D0 F8 4C C7 08   38
08E1 : C9 1D D0 03 4C 7B 09 C9   26
08E9 : 9D D0 03 4C 8D 09 C9 14   A9
08F1 : D0 03 4C E0 09 C9 94 D0   45
08F9 : 03 4C BB 09 C9 11 D0 03   A1
0901 : 4C 98 09 C9 91 D0 03 4C   59
0909 : AB 09 C9 8C D0 03 4C 0E   AF
0911 : 0E C9 85 D0 08 A9 01 8D   6C
0919 : 63 0D 4C D1 08 C9 86 D0   DB
0921 : 03 4C 80 0C C9 87 D0 03   0E
0929 : 4C A7 0C C9 88 D0 03 4C   39
0931 : D6 0C C9 8A D0 03 4C 1A   5B
0939 : 0F C9 8B D0 06 20 E7 0D   45
0941 : 4C AC 0A C9 89 D0 03 4C   63
0949 : D2 0E C9 00 D0 0A 68 68   65
0951 : A9 FF 8D 41 08 4C 2A 08   21
0959 : C9 01 D0 0A 68 68 A9 17   B7
0961 : 8D 41 08 4C 2A 08 C9 12   49
0969 : F0 10 C9 92 F0 0C C9 20   FD
0971 : 90 19 C9 80 90 04 C9 A0   A2
0979 : 90 11 20 D2 FF A9 00 85   4D
0981 : D4 A4 D3 CC A9 0A D0 03   6A
0989 : C6 D3 60 60 A4 D3 CC AA   CE
0991 : 0A D0 01 60 4C D2 FF AC   04
0999 : A7 0A CC A8 0A F0 08 EE   B4
09A1 : A7 0A A9 11 4C D2 FF 4C   CE
09A9 : 07 0B AC A7 0A F0 08 CE   3C
09B1 : A7 0A A9 91 4C D2 FF 4C   EE
09B9 : 2A 0B AC A9 0A 88 C4 D3   69
09C1 : F0 14 20 24 EA 88 B1 D1   A5
09C9 : C8 91 D1 88 B1 F3 C8 91   E0
09D1 : F3 88 C4 D3 D0 EF A9 20   27
09D9 : 91 D1 A9 00 85 D8 60 A4   A7
09E1 : D3 CC AA 0A D0 01 60 20   DD
09E9 : 24 EA B1 D1 88 91 D1 C8   17
09F1 : B1 F3 88 91 F3 C8 C8 CC   32
09F9 : A9 0A D0 EE 88 A9 20 91   33
0A01 : D1 C6 D3 60 8D 7D 0A 8C   3C
0A09 : AB 0A 8C 62 0A 8E 39 0A   37
0A11 : 38 20 F0 FF C8 8C AA 0A   45
0A19 : A9 B0 20 D2 FF A9 C0 20   0D
0A21 : 76 0A A9 AE 20 D2 FF A9   C8
0A29 : 0D 20 D2 FF AC AA 0A 88   54
0A31 : 84 D3 A9 DD 20 D2 FF A9   B1
0A39 : 2E 20 76 0A A9 DD 20 D2   06
0A41 : FF CE AB 0A D0 E1 A9 0D   B0
0A49 : 20 D2 FF AC AA 0A 88 84   8E
0A51 : D3 A9 AD 20 D2 FF A9 C0   BD
0A59 : 20 76 0A A9 BD 20 D2 FF   94
0A61 : A0 00 A9 91 20 D2 FF CE   D4
0A69 : 62 0A D0 F8 AD AA 0A 85   87
0A71 : D3 AD 7D 0A 60 A0 00 20   07
0A79 : D2 FF C8 C0 00 D0 F8 60   C0
0A81 : 8D 8F 0A 8C 90 0A A5 D3   82
0A89 : 8D 9B 0A A0 00 B9 FF FF   48
0A91 : F0 13 20 D2 FF C9 0D D0   91
0A99 : 04 A9 00 85 D3 C8 D0 ED   C5
0AA1 : EE 90 0A D0 E8 60 00 00   06
0AA9 : 00 00 00 AD 05 0B 85 FB   16
0AB1 : AD 06 0B 85 FC A9 17 85   59
0AB9 : 02 38 20 F0 FF 8A 48 98   A4
0AC1 : 48 A9 13 20 D2 FF A9 0D   95
0AC9 : 20 D2 FF A9 1D 20 D2 FF   A5
0AD1 : A0 00 B1 FB 84 22 A2 05   4B
0AD9 : 20 13 EA 20 B6 E6 A4 22   BB
0AE1 : C8 C0 18 D0 ED A5 FB 18   56
0AE9 : 69 40 85 FB A5 FC 69 00   3B
0AF1 : 85 FC C6 02 D0 D0 68 A8   6D
0AF9 : 68 AA 18 20 F0 FF A9 00   76
0B01 : 8D 62 0D 60 00 20 AD 05   D0
0B09 : 0B C9 40 D0 08 AD 06 0B   3F
0B11 : C9 9A D0 01 60 AD 05 0B   19
0B19 : 18 69 40 8D 05 0B AD 06   13
0B21 : 0B 69 00 8D 06 0B 4C AC   D6
0B29 : 0A AD 05 0B C9 00 D0 08   9D
0B31 : AD 06 0B C9 20 D0 01 60   2B
0B39 : AD 05 0B 38 E9 40 8D 05   14
0B41 : 0B AD 06 0B E9 00 8D 06   E7
0B49 : 0B 4C AC 0A A9 00 8D A7   07
0B51 : 0B AD A7 0A 0A 2E A7 0B   25
0B59 : 0A 2E A7 0B 0A 2E A7 0B   8C
0B61 : 0A 2E A7 0B 0A 2E A7 0B   94
0B69 : 0A 2E A7 0B 18 6D 05 0B   ED
0B71 : 8D A6 0B 85 FB AD A7 0B   A7
0B79 : 6D 06 0B 8D A7 0B 85 FC   41
0B81 : 60 AD 63 0D F0 [illegible] A9 00   50
0B89 : 8D 63 0D 68 68 20 AD 0B   6D
0B91 : 4C AC 0A AD 62 0D F0 01   C0
0B99 : 60 20 4D 0B A2 17 AC A9   A7
0BA1 : 0A 88 B1 D1 9D FF FF 88   81
0BA9 : CA 10 F7 60 A4 D3 88 D0   32
0BB1 : 02 A0 18 8C 7F 0C A9 00   9A
0BB9 : 8D 7D 0C 8D 7E 0C 20 4D   1D
0BC1 : 0B E6 D1 D0 02 E6 D2 4C   09
0BC9 : 52 0C A0 00 B1 D1 C9 3F   99
0BD1 : F0 04 D1 FB D0 7B C8 CC   5D
0BD9 : 7F 0C D0 F0 A5 FC C9 9A   4F
0BE1 : 90 50 D0 06 A5 FB C9 40   70
0BE9 : 90 48 A9 40 8D 05 0B A9   90
0BF1 : 9A 8D 05 0B A5 FB 38 E9   E3
0BF9 : 40 85 FD A5 FC E9 94 85   AC
0C01 : FE 46 FE 66 FD 46 FE 66   8A
0C09 : FD 46 FE 66 FD 46 FE 66   91
0C11 : FD 46 FE 66 FD 46 FE 66   99
0C19 : FD A5 FD 8D A7 0A A8 A9   DB
0C21 : 13 20 D2 FF A9 1D 20 D2   A2
0C29 : FF A9 11 20 D2 FF 88 D0   36
0C31 : FA 60 A5 FB 8D 05 0B A5   BD
0C39 : FC 8D 06 0B A9 13 20 D2   38
0C41 : FF A9 11 20 D2 FF A9 1D   6B
0C49 : 20 D2 FF A9 00 8D A7 0A   26
0C51 : 60 EE 7D 0C D0 03 EE 7E   E7
0C59 : 0C AD 7D 0C F0 07 AD 7E   18
0C61 : 0C C9 02 F0 16 A5 FB 18   9F
0C69 : 69 40 85 FB A5 FC 69 00   BB
0C71 : C9 A0 D0 02 A9 20 85 FC   AA
0C79 : 4C CB 0B 60 00 00 00 AD   D5
0C81 ; 62 0D F0 01 60 20 4D 0B   18
0C89 : A5 FB 8D D3 0C A5 FC 8D   07
0C91 : D4 0C A0 18 B1 FB 29 01   38
0C99 : D0 06 C8 C8 C0 40 D0 F4   F3
0CA1 : 8C D5 0C 4C B4 08 AD 62   AB
0CA9 : 0D F0 01 60 20 4D 0B AD   6F
0CB1 : D3 0C 85 FD AD D4 0C 85   68
0CB9 : FE AC D5 0C C0 40 D0 02   5A
0CC1 : A0 18 A5 FB 91 FD C8 A5   CE
0CC9 : FC 91 FD C8 8C D5 0C 4C   66
0CD1 : B4 08 00 20 18 20 B2 0B   F1
0CD9 : A5 FB 85 FD A5 FC 85 FE   F3
0CE1 : A9 18 8D 61 0D A2 01 A0   51
0CE9 : 00 18 20 F0 FF 20 2E 0D   EF
0CF1 : 20 4D 0D 20 4D 0D AC 61   B2
0CF9 : 0D C0 40 F0 1F B1 FD 85   17
0D01 : FB 29 01 48 C8 B1 FD 85   F7
0D09 : FC C8 8C 61 0D 68 D0 06   1C
0D11 : 20 2E 0D 4C F7 0C 20 4D   10
0D19 : 0D 4C F7 0C A9 01 8D 62   69
0D21 : 0D A2 03 8E A7 0A E8 A0   C2
0D29 : 01 18 4C F0 FF A9 1D 20   69
0D31 : D2 FF A0 00 B1 FB 84 02   3C
0D39 : A2 01 20 13 EA 20 B6 E6   1F
0D41 : A4 02 C8 C0 18 D0 ED A9   44
0D49 : 0D 4C D2 FF A9 1D 20 D2   DA
0D51 : FF A0 18 A9 20 20 D2 FF   2A
0D59 : 88 D0 FA A9 0D 4C D2 FF   BC
0D61 : 00 00 00 20 20 20 20 20   29
0D69 : 20 20 20 20 20 20 20 20   69
0D71 : 20 20 20 20 20 20 20 20   71
0D79 : 20 20 20 01 01 01 01 01   F0
0D81 : 01 01 01 01 01 01 01 01   81
0D89 : 01 01 01 01 01 01 01 01   89
0D91 : 01 01 01 01 01 01 01 01   91
0D99 : 01 01 01 01 01 01 01 01   99
0DA1 : 01 01 01 A4 D3 88 8C E6   79
0DA9 : 0D C6 D3 A4 D3 D0 05 E6   48
0DB1 : D3 4C AC 0A B1 D1 85 D7   86
0DB9 : 29 3F 06 D7 24 D7 10 02   44
0DC1 : 09 80 70 02 09 40 88 99   4F
0DC9 : CE 0D 4C AA 0D 53 49 47   A5
0DD1 : 4E 53 20 20 20 20 20 20   99
0DD9 : 20 20 20 20 20 20 20 20   D9
0DE1 : 20 20 20 20 20 05 20 A4   11
0DE9 : 0D AD E6 0D D0 01 60 AC   18
0DF1 : CE 0D C0 24 D0 03 4C 57   FF
0DF9 : 0F A2 CE A0 0D 20 BD FF   EA
0E01 : A9 08 85 BA A9 01 85 B9   93
0E09 : A9 00 4C D5 FF 20 A4 0D   2E
0E11 : AD E6 0D D0 01 60 A2 CE   CA
0E19 : A0 0D 20 BD FF 20 4D 0B   4C
0E21 : A6 FB A4 FC A9 08 85 BA   F4
0E29 : A9 00 85 FD A9 20 85 FE   A3
0E31 : A9 FD 4C D8 FF A9 93 20   E3
0E39 : D2 FF A2 00 A0 1A 18 20   2F
0E41 : F0 FF A9 0B A0 0A A2 20   22
0E49 : 20 05 0A A2 0D A0 1A 18   31
0E51 : 20 F0 FF A9 0B A0 0A A2   41
0E59 : 20 20 05 0A A2 02 A0 1B   FF
0E61 : 18 20 F0 FF A9 6C A0 0E   62
0E69 : 4C 81 0A 46 31 20 53 45   AD
0E71 : 41 52 43 48 0D 0D 46 33   6E
0E79 : 20 42 41 53 49 53 0D 0D   F3
0E81 : 46 35 20 53 54 52 45 45   4C
0E89 : 54 0D 0D 46 37 20 44 52   9A
0E91 : 49 56 45 0D 0D 0D 0D 9D   A1
0E99 : 12 20 D4 C1 D8 C9 2D D3   61
0EA1 : C3 C8 CF CF CC 20 92 0D   E8
0EA9 : 0D 0D 46 32 20 49 4E 53   41
0EB1 : 45 52 54 0D 0D 46 34 20   EA
0EB9 : 44 45 4C 45 54 45 0D 0D   19
0EC1 : 46 36 20 4C 4F 41 44 0D   DE
0EC9 : 0D 46 38 20 53 41 56 45   2F
0ED1 : 00 AD 62 0D F0 01 60 20   BB
0ED9 : 4D 0B A9 C0 85 FB A9 9F   4C
0EE1 : 85 FC A9 80 85 FD A9 9F   8D
0EE9 : 85 FE 20 4D 0F A5 FD 85   C0
0EF1 : FB A5 FE 85 FC A5 FD 38   94
0EF9 : E9 40 85 FD A5 FE E9 00   1D
0F01 : 85 FE A5 FB CD A6 0B D0   CE
0F09 : E1 A5 FC CD A7 0B D0 DA   81
0F11 : 20 D1 08 20 82 0B 4C AC   2B
0F19 : 0A AD 62 0D F0 01 60 20   0D
0F21 : 4D 0B A5 FB 18 69 40 85   B5
0F29 : FD A5 FC 69 00 85 FE 20   CE
0F31 : 4D 0F A5 FD 85 FB A5 FE   FC
0F39 : 85 FC C9 A0 D0 E4 A0 3F   F8
0F41 : B9 64 0D 99 C0 9F 88 10   EE
0F49 : F7 4C AC 0A A0 3F B1 FD   99
0F51 : 91 FB 88 10 F9 60 A9 00   4D
0F59 : 8D 89 02 A9 14 8D 09 10   53
0F61 : A9 93 20 D2 FF A9 24 85   1F
0F69 : FB A9 FB 85 BB A9 00 85   FC
0F71 : BC A9 01 85 B7 A9 08 85   E7
```

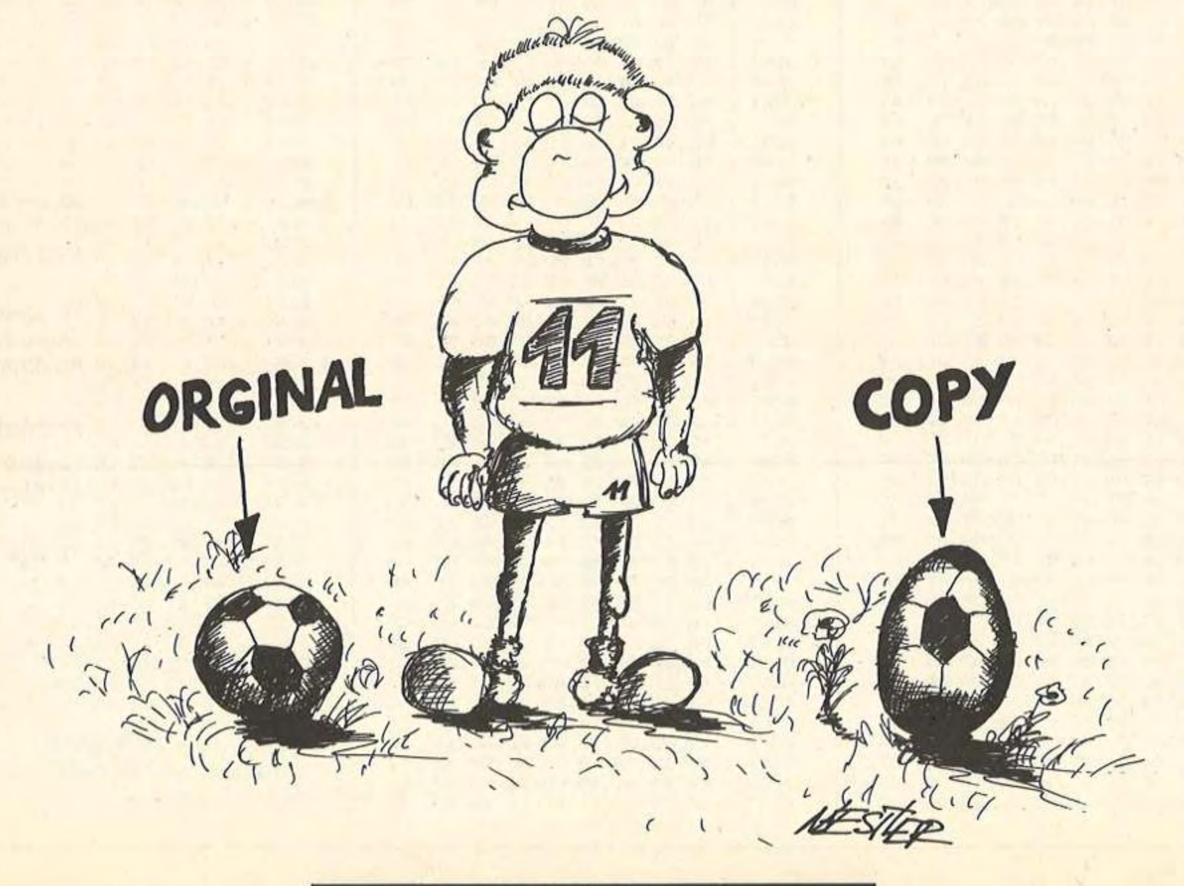
Listing 12. »Taxi-school«, die etwas andere Dateiverwaltung (Fortsetzung)

```
0F79 : BA A9 60 85 B9 20 D5 F3  AC
0F81 : A5 BA 20 B4 FF A5 B9 20  76
0F89 : 96 FF A9 00 85 90 A0 03  EF
0F91 : 84 FB 20 A5 FF 85 FC A4  39
0F99 : 90 D0 32 20 A5 FF A4 90  30
0FA1 : D0 2B A4 FB 88 D0 E9 A6  B3
0FA9 : FC 20 CD BD A9 20 20 D2  A2
0FB1 : FF 20 A5 FF A6 90 D0 15  86
0FB9 : AA F0 06 20 D2 FF 4C B2  25
0FC1 : 0F A9 0D 20 D2 FF 20 E0  5C
0FC9 : 0F A0 02 D0 C3 20 42 F6  F7
0FD1 : A0 00 20 EC 0F A9 0A 8D  98
0FD9 : 89 02 68 68 4C 55 08 CE  B8
0FE1 : 09 10 D0 1D A9 14 8D 09  4E
0FE9 : 10 A0 00 B9 0A 10 F0 06  72
0FF1 : 20 D2 FF C8 D0 F5 A5 CB  7E
0FF9 : C9 40 F0 FA A9 93 20 D2  DB
1001 : FF A5 CB C9 40 D0 FA 60  36
1009 : 00 0D 54 41 53 54 45 20  FA
1011 : 44 52 55 45 43 4B 45 4E  BD
1019 : 00 E8 D0 E5 60 00 00 00  84
```

Listing 12. »Taxi-school« (Schluß)

```
NAME : SIGNS              C000 C800
-----------------------------------
C000 : 00 66 00 66 66 66 3E 00  93
C008 : 00 00 3C 06 3E 66 3E 00  E8
C010 : 00 60 60 7C 66 66 7C 00  74
C018 : 00 00 3C 60 60 60 3C 00  2D
C020 : 00 06 06 3E 66 66 3E 00  FF
C028 : 00 00 3C 66 7E 60 3C 00  E0
C030 : 00 0E 18 3E 18 18 18 00  A8
C038 : 00 00 3E 66 66 3E 06 7C  FE
C040 : 00 60 60 7C 66 66 66 00  4B
C048 : 00 18 00 38 18 18 3C 00  8F
C050 : 00 06 00 06 06 06 06 3C  35
C058 : 00 60 60 6C 78 6C 66 00  B2
C060 : 00 38 18 18 18 18 3C 00  B9
C068 : 00 00 66 7F 7F 6B 63 00  D3
C070 : 00 00 7C 66 66 66 66 00  8F
C078 : 00 00 3C 66 66 66 3C 00  DF
C080 : 00 00 7C 66 66 7C 60 60  F9
C088 : 00 00 3E 66 66 3E 06 06  61
C090 : 00 00 7C 66 60 60 60 00  07
C098 : 00 00 3E 60 3C 06 7C 00  1A
C0A0 : 00 18 7E 18 18 18 0E 00  C9
C0A8 : 00 00 66 66 66 66 3E 00  A1
C0B0 : 00 00 66 66 66 3C 18 00  BF
C0B8 : 00 00 63 6B 7F 3E 36 00  C1
C0C0 : 00 00 66 3C 18 3C 66 00  DE
C0C8 : 00 00 66 66 66 3E 0C 78  AB
C0D0 : 00 00 7E 0C 18 30 7E 00  EE
C0D8 : 66 00 3C 66 66 66 3C 00  A5
C0E0 : 3C 66 66 6C 66 66 6C 60  83
C0E8 : 66 00 18 3C 66 7E 66 00  D0
C0F0 : 18 3C 7E 18 18 18 18 00  6C
C0F8 : 00 10 30 7F 7F 30 10 00  B6
C100 : 00 00 00 00 00 00 00 00  01
C108 : 18 18 18 18 00 00 18 00  96
C110 : 66 66 66 00 00 00 00 00  43
C118 : 66 66 FF 66 FF 66 66 00  4B
C120 : 18 3E 60 3C 06 7C 18 00  9C
C128 : 62 66 0C 18 30 66 46 00  13
C130 : 3C 66 3C 38 67 66 3F 00  5C
C138 : 06 0C 18 00 00 00 00 00  4B
C140 : 0C 18 30 30 30 18 0C 00  5E
C148 : 30 18 0C 0C 0C 18 30 00  4B
C150 : 00 66 3C FF 3C 66 00 00  89
C158 : 00 18 18 7E 18 18 00 00  7D
C160 : 00 00 00 00 00 18 18 30  E2
C168 : 00 00 00 7E 00 00 00 00  38
C170 : 00 00 00 00 00 18 18 00  92
C178 : 00 03 06 0C 18 30 60 00  82
C180 : 3C 66 6E 76 66 66 3C 00  E4
C188 : 18 18 38 18 18 18 7E 00  FA
C190 : 3C 66 06 0C 30 60 7E 00  02
C198 : 3C 66 06 1C 06 66 3C 00  91
C1A0 : 06 0E 1E 66 7F 06 06 00  42
C1A8 : 7E 60 7C 06 06 66 3C 00  BB
C1B0 : 3C 66 60 7C 66 66 3C 00  52
C1B8 : 7E 66 0C 18 18 18 18 00  12
C1C0 : 3C 66 66 3C 66 66 3C 00  DB
C1C8 : 3C 66 66 3E 06 66 3C 00  1D
C1D0 : 00 66 00 3C 66 66 3C 00  16
C1D8 : 00 66 00 3C 06 7E 3C 00  D8
C1E0 : 00 00 18 00 00 18 18 30  68
C1E8 : 00 00 7E 00 7E 00 00 00  70
C1F0 : 00 00 18 00 00 18 00 00  B7
C1F8 : 3C 66 06 0C 18 00 18 00  4C
C200 : 00 00 00 FF FF 00 00 00  00
C208 : 18 3C 66 7E 66 66 66 00  DB
C210 : 7C 66 66 7C 66 66 7C 00  74
C218 : 3C 66 60 60 60 66 3C 00  D6
C220 : 78 6C 66 66 66 6C 78 00  E0
C228 : 7E 60 60 78 60 60 7E 00  00
C230 : 7E 60 60 78 60 60 60 00  90
C238 : 3C 66 60 6E 66 [illegible] [illegible] 00  18
C240 : 66 66 66 7E 66 66 66 00  76
C248 : 3C 18 18 18 18 18 3C 00  CD
C250 : 1E 0C 0C 0C 0C 6C 38 00  FE
C258 : 66 6C 78 70 78 6C 66 00  A5
C260 : 60 60 60 60 60 60 7E 00  17
C268 : 63 77 7F 6B 63 63 63 00  B3
C270 : 66 76 7E 7E 6E 66 66 00  34
C278 : 3C 66 66 66 66 66 3C 00  D8
C280 : 7C 66 66 7C 60 60 60 00  E3
C288 : 3C 66 66 66 66 3C 0E 00  DE
C290 : 7C 66 66 7C 78 6C 66 00  ED
C298 : 3C 66 60 3C 06 66 3C 00  2C
C2A0 : 7E 18 18 18 18 18 18 00  D6
C2A8 : 66 66 66 66 66 66 3C 00  32
C2B0 : 66 66 66 66 66 3C 18 00  58
C2B8 : 63 63 63 6B 7F 77 63 00  54
C2C0 : 66 66 3C 18 3C 66 66 00  FC
C2C8 : 66 66 66 3C 18 18 18 00  25
C2D0 : 7E 06 0C 18 30 60 7E 00  57
C2D8 : 18 18 18 7E 18 18 18 18  A5
C2E0 : 00 3C 7E 42 42 7E 3C 00  EF
C2E8 : 18 18 18 18 18 18 18 18  E8
C2F0 : 10 10 38 FE 6C 6C C6 00  3B
C2F8 : DA DA DA DE 8C 8C 8C 00  31
C300 : 00 00 00 00 00 00 00 00  01
C308 : FF C3 A5 99 99 A5 C3 FF  5B
C310 : FF C3 FF C3 C3 C3 81 FF  C9
C318 : F0 98 96 F6 F4 F2 FE 00  BC
C320 : 00 00 00 00 00 00 00 FF  20
C328 : 01 03 07 0F 1F 3F 7F FF  39
C330 : CC CC 33 33 CC CC 33 33  FC
C338 : 6C FE C6 C6 C6 C6 FE 00  4C
C340 : 10 38 6C C6 C6 C6 FE 00  FF
C348 : 3C 60 3C 66 3C 06 3C 00  75
C350 : 01 03 06 0C 18 30 60 C0  DC
C358 : FE 06 FE C0 FE 00 38 00  02
C360 : 7E E7 C3 C3 C3 C3 C3 FF  A4
C368 : 18 18 18 1F 1F 00 00 00  68
C370 : 00 00 00 F8 F8 18 18 18  70
C378 : FC C6 C6 FC C0 FE FE 00  28
C380 : 00 00 00 1F 1F 18 18 18  A8
C388 : 00 0C FE C3 C3 FE 0C 00  2B
C390 : C3 66 00 18 18 00 66 C3  2C
C398 : FE 6C 6C 6C 6C 6C 6C 00  51
C3A0 : 0C F2 F2 CC C0 C0 C0 00  91
C3A8 : E1 E1 E1 E1 E1 E1 E1 E1  A7
C3B0 : 87 87 87 87 87 87 87 87  AF
C3B8 : FF FF FF 00 00 00 00 FF  B7
C3C0 : FF 99 99 99 99 99 C3 FF  9B
C3C8 : 3C 66 C3 C3 C3 C3 66 3C  0D
C3D0 : 66 00 66 66 66 66 3C 00  27
```

Listing 13. Der geänderte Zeichensatz zu »Taxi-school«



```
C3D8 : F0 66 66 7E 00 66 66 00  32
C3E0 : FF C3 C3 E7 E7 E7 E7 FF  0C
C3E8 : 18 18 18 F8 F8 00 00 00  C1
C3F0 : 18 18 00 18 18 00 18 18  2A
C3F8 : 03 06 0C 78 F0 F8 70 00  A9
C400 : FF 99 FF 99 99 99 C1 FF  6C
C408 : FF FF C3 F9 C1 99 C1 FF  27
C410 : FF 9F 9F 83 99 99 83 FF  AC
C418 : FF FF C3 9F 9F 9F C3 FF  02
C420 : FF F9 F9 C1 99 99 C1 FF  40
C428 : FF FF C3 99 81 9F C3 FF  6F
C430 : FF F1 E7 C1 E7 E7 E7 FF  B7
C438 : FF FF C1 99 99 C1 F9 83  71
C440 : FF 9F 9F 83 99 99 99 FF  34
C448 : FF E7 FF C7 E7 E7 C3 FF  00
C450 : FF F9 FF F9 F9 F9 F9 C3  6A
C458 : FF 9F 9F 93 87 93 99 FF  FD
C460 : FF C7 E7 E7 E7 E7 C3 FF  06
C468 : FF FF 99 80 80 94 9C FF  FC
C470 : FF FF 83 99 99 99 99 FF  50
C478 : FF FF C3 99 99 99 C3 FF  10
C480 : FF FF 83 99 99 83 9F 9F  06
C488 : FF FF C1 99 99 C1 F9 F9  AE
C490 : FF FF 83 99 9F 9F 9F FF  18
C498 : FF FF C1 9F C3 F9 83 FF  15
C4A0 : FF E7 81 E7 E7 E7 F1 FF  76
C4A8 : FF FF 99 99 99 99 C1 FF  AE
C4B0 : FF FF 99 99 99 C3 E7 FF  A0
C4B8 : FF FF 9C 94 80 C1 C9 FF  AE
C4C0 : FF FF 99 C3 E7 C3 99 FF  A1
C4C8 : FF FF 99 99 99 C1 F3 87  E7
C4D0 : FF FF 81 F3 E7 CF 81 FF  B1
C4D8 : 99 FF C3 99 99 99 C3 FF  0A
C4E0 : C3 99 99 93 99 99 93 9F  3D
C4E8 : 99 FF E7 C3 99 81 99 FF  FF
C4F0 : E7 E7 C3 00 99 00 3C 3C  BF
C4F8 : FF EF CF 80 80 CF EF FF  39
C500 : FF FF FF FF FF FF FF FF  FF
C508 : E7 E7 E7 E7 FF FF E7 FF  79
C510 : 99 99 99 FF FF FF FF FF  DC
C518 : 99 99 00 99 00 99 99 FF  E4
C520 : E7 C1 9F C3 F9 83 E7 FF  A3
C528 : 9D 99 F3 E7 CF 99 B9 FF  3C
C530 : C3 99 C3 C7 98 99 C0 FF  03
C538 : F9 F3 E7 FF FF FF FF FF  25
C540 : F3 E7 CF CF CF E7 F3 FF  21
C548 : CF E7 F3 F3 F3 E7 CF FF  44
C550 : FF 99 C3 00 C3 99 FF FF  16
C558 : FF E7 E7 81 81 E7 E7 FF  6C
C560 : FF FF FF FF FF E7 E7 CF  DD
C568 : FF FF FF 81 FF FF FF FF  97
C570 : FF FF FF FF FF E7 E7 FF  4D
C578 : FF FC F9 F3 E7 CF 9F FF  6E
C580 : C3 99 91 89 99 99 C3 FF  1B
C588 : E7 E7 C7 E7 E7 E7 81 FF  15
C590 : C3 99 F9 F3 CF 9F 81 FF  1D
C598 : C3 99 F9 E3 F9 99 C3 FF  9E
C5A0 : F9 F1 E1 99 80 F9 F9 FF  FD
C5A8 : 81 9F 83 F9 F9 99 C3 FF  94
C5B0 : C3 99 9F 83 99 99 C3 FF  0E
C5B8 : 81 99 F3 E7 E7 E7 E7 FF  5D
C5C0 : C3 99 99 C3 99 99 C3 FF  A4
C5C8 : C3 99 99 C1 F9 99 C3 FF  72
C5D0 : FF 99 FF C3 99 99 C3 FF  8A
C5D8 : FF 99 FF C3 F9 81 C3 FF  D7
C5E0 : FF FF E7 FF FF E7 E7 CF  57
C5E8 : FF FF 81 FF 81 FF FF FF  5F
C5F0 : FF FF E7 FF FF E7 FF FF  28
C5F8 : C3 99 F9 F3 E7 FF E7 FF  A3
C600 : FF FF FF 00 00 FF FF FF  FF
C608 : E7 C3 99 81 99 99 99 FF  34
C610 : 83 99 99 83 99 99 83 FF  AB
C618 : C3 99 9F 9F 9F 99 C3 FF  59
C620 : 87 93 99 99 99 93 87 FF  5F
C628 : 81 9F 9F 87 9F 9F 81 FF  4F
C630 : 81 9F 9F 87 9F 9F 9F FF  CF
C638 : C3 99 9F 91 99 99 C3 FF  57
C640 : 99 99 99 81 99 99 99 FF  09
C648 : C3 E7 E7 E7 E7 E7 C3 FF  C2
C650 : E1 F3 F3 F3 F3 93 C7 FF  A1
C658 : 99 93 87 8F 87 93 99 FF  0A
C660 : 9F 9F 9F 9F 9F 9F 81 FF  A8
C668 : 9C 88 80 94 9C 9C 9C FF  1C
C670 : 99 89 81 81 91 99 99 FF  AB
C678 : C3 99 99 99 99 99 C3 FF  17
C680 : 83 99 99 83 9F 9F 9F FF  1C
C688 : C3 99 99 99 99 C3 F1 FF  31
C690 : 83 99 99 83 87 93 99 FF  32
C698 : C3 99 9F C3 F9 99 C3 FF  04
C6A0 : 81 E7 E7 E7 E7 E7 E7 FF  69
C6A8 : 99 99 99 99 99 99 C3 FF  1D
C6B0 : 99 99 99 99 99 C3 E7 FF  07
C6B8 : 9C 9C 9C 94 80 88 9C FF  1B
C6C0 : 99 99 C3 E7 C3 99 99 FF  83
C6C8 : 99 99 99 C3 E7 E7 E7 FF  6A
C6D0 : 81 F9 F3 E7 CF 9F 81 FF  48
C6D8 : E7 E7 81 E7 E7 E7 E7 E7  3D
C6E0 : 3F 3F CF CF 3F 3F CF CF  79
C6E8 : E7 E7 E7 E7 E7 E7 E7 E7  E7
C6F0 : 07 73 74 04 05 06 01 FF  D3
C6F8 : CC 66 33 99 CC 66 33 99  F7
C700 : FF FF FF FF FF FF FF FF  FF
C708 : 0F 0F 0F 0F 0F 0F 0F 0F  08
C710 : FF FF FF FF 00 00 00 00  0F
C718 : 00 FF FF FF FF FF FF FF  18
C720 : FF FF FF FF FF FF FF 00  1F
C728 : 3F 3F 3F 3F 3F 3F 3F 3F  28
C730 : 33 33 CC CC 33 33 CC CC  63
C738 : FC FC FC FC FC FC FC FC  37
C740 : E7 C3 99 3C BD BD BD 81  BA
C748 : C3 9F C3 99 C3 F9 C3 FF  1A
C750 : FC FC FC FC FC FC FC FC  4F
C758 : E7 E7 E7 E0 E0 E7 E7 E7  05
C760 : FF FF FF FF F0 F0 F0 F0  9B
C768 : E7 E7 E7 E0 E0 FF FF FF  67
C770 : FF FF FF 07 07 E7 E7 E7  6F
C778 : FF FF FF FF FF FF 00 00  77
C780 : FF FF FF E0 E0 E7 E7 E7  57
C788 : E7 E7 E7 00 00 FF FF FF  5D
C790 : FF FF FF 00 00 E7 E7 E7  3D
C798 : E7 E7 E7 07 07 E7 E7 E7  6D
C7A0 : 3F 3F 3F 3F 3F 3F 3F 3F  A0
C7A8 : 1F 1F 1F 1F 1F 1F 1F 1F  A8
C7B0 : F8 F8 F8 F8 F8 F8 F8 F8  AF
C7B8 : 00 00 FF FF FF FF FF FF  B8
C7C0 : 00 00 00 FF FF FF FF FF  C0
C7C8 : FF FF FF FF FF 00 00 00  C7
C7D0 : 99 FF 99 99 99 99 C3 FF  78
C7D8 : FF FF FF FF 0F 0F 0F 0F  9A
C7E0 : F0 F0 F0 F0 FF FF FF FF  A2
C7E8 : E7 E7 E7 07 07 FF FF FF  0E
C7F0 : 0F 0F 0F 0F FF FF FF FF  2D
C7F8 : 0F 0F 0F 0F F0 F0 F0 F0  71
```

Listing 13. Zeichensatz (Schluß)

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Chefredakteur: Michael Scharfenberger
Stellv. Chefredakteur: Albert Absmeier
Koordination: Georg Klinge
Redaktion: Achim Hübner (ah), Karsten Schramm (ks), Herbert Buckel (bj), Dieter Mayer (dm), Markus Ohnesorg (og), Norbert Jungmann (nj), Gerd Donaubauer (do), Thomas Röder (tr)

Titelfoto: Jens Jancke

Titelgestaltung: Heinz Rauner Grafik-Design

Layout:
Leo Eder (Ltg.), Sigrid Kowalewski (Cheflayouterin), Rolf Raß, Katja Milles

Produktionsleiter: Klaus Buck

Auslandsrepräsentation:
Schweiz: Markt & Technik Vertriebs AG, Kollerstr. 3, CH-6300 Zug, Tel. 042-41 56 56, Telex: 862 329
USA: M&T Publishing Inc.; 501 Galveston Drive Redwood City, CA 94063 Telefon: (4 15) 366-3600

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten werden, so muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt & Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt & Technik Verlag AG verlegten Publikationen und dazu, daß Markt & Technik Verlag AG Geräte und Bauteile nach der Bauanleitung herstellen läßt und vertreibt oder durch Dritte vertreiben läßt. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Marketingleiter: Hans Hörl (114)

Vertriebsleiter: Helmut Grünfeldt (189)

Anzeigenverwaltung und Disposition: Michaela Hörl

Verlagsleiter M&T-Buchverlag: Günther Frank (212)

Druck: SOV St. Otto-Verlag GmbH, Laubanger 23, 8600 Bamberg

Preis: Das Einzelheft kostet DM 14,–

Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: Pegasus Buch- und Zeitschriften-Vertriebs GmbH, Hauptstätter Straße 96, 7000 Stuttgart 1, Telefon (07 11) 6 48 30

Verantwortlich:
Für redaktionellen Teil: Michael Scharfenberger
Für Anzeigen: Britta Fiebig

Redaktions-Direktor: Michael M. Pauly

Vorstand: Carl-Franz von Quadt, Otmar Weber

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen:
Markt & Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 46 13-0, Telex 5-22 052




Bücher zum Commodore 64


PROTEXT
TOP
ASS
64er-online.de
64er-online.net